

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。



微信公众平台 开发与案例分析

张思凯 著

在这个O2O、大数据、互联网+盛行的大环境下，微信公众平台的开放性与包容性使各行各业以群雄逐鹿之势投入其中

本书来自于作者近三年来微信开发的经验总结，图书结合视频教程，使您充分理解微信公众平台的工作原理，定制自己的微信开发SDK，在微信开发的道路上更轻松愉悦



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

作者简介



张思凯

资深微信开发者，多年微信开发工作经验，2013年年初接触微信公众平台开发，先后在汽车、餐饮、房地产等多个热门行业进行微信O2O项目的开发。2015年年初辞职创业，成立武汉微企卡科技有限公司。

作者微信公众号



微信公众平台 开发与案例分析

张思凯 著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

移动互联网时代,信息瞬息万变,微信公众号异军突起,众开发者也纷纷加入。本书来源于笔者近三年微信公众平台开发经验的总结,以C#为技术基础,详细讲解微信公众平台的所有基础接口和绝大部分高级接口的调用、代码编写以及使用场景。从公众平台的工作原理到基础的开发与调试环境的搭建,再到基础服务接口的使用,最后在讲解各个高级接口调用的同时,结合实战案例,使读者对各个接口的调用以及使用场景有个充分的认识。

本书适合.NET平台下,有一定C#开发经验的开发者阅读,也可作为技术培训机构的参考教材。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

微信公众平台开发与案例分析/张思凯著. —北京:电子工业出版社, 2015.9
ISBN 978-7-121-27116-8

I. ①微… II. ①张… III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆CIP数据核字(2015)第214339号

策划编辑:陈晓猛

责任编辑:李云静

印 刷:三河市双峰印刷装订有限公司

装 订:三河市双峰印刷装订有限公司

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编 100036

开 本:787×980 1/16 印张:32 字数:674千字

版 次:2015年9月第1版

印 次:2015年9月第1次印刷

印 数:3000册 定价:79.00元(含DVD光盘1张)

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010)88254888。

质量投诉请发邮件至 zltts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010)88258888。

前言



为什么写这么一本书

2013 年的 5 月份，我和 BOSS 一起去参加了一个会议。到现在我都不清楚那场会议的主办方和会议的目的，估计应该属于会销的一种。作为一个“菜鸟”，听着那些“高大上”的营销策略、营销方式，让我觉得自己真的是“too young too simple, sometimes naive”；不过最大的收获就是领悟到了微信公众平台的魅力。只是一个简单的关键词回复就让我心动不已，从此开始了自己漫长的微信研究之路。开发入门是痛苦的，当时甚至不知道字典序是什么，开始到处找 demo，最后在 CSDN 花费了 10 个积分下载了一个 demo，然后就是熬夜部署测试。由于在这之前并没有接触过类似的接口开发，因此走了不少弯路。后来对着官方文档和 demo 一行行代码进行分析，最终成功接入。万事开头难，入门了才发现：“一入此行深似海，从此妹子是路人。”那时几近疯狂，每每陪我家妹子逛街时发现了某个商家的二维码，都会熟练地拿出配置极低的某款手机→打开微信→扫一扫。只为能在开发微信的过程中，获得更多的素材与灵感。

2014 年 10 月，我进入了一家创业公司，这家公司的员工是清一色的“90 后”，注定大家求知欲比较旺盛（主要还是因为“too young too simple”）。技术部的“90 后”领导决定每周组织大家学习交流，我很是兴奋：一是在这之前带实习生的过程中感受到的成就感，二是真心希望在公司里不止我一个人懂微信开发。为此我精心准备了几天的时间，终于到了交流的时间，可是大家的表现却让我的心凉了半截：我唾沫横飞地讲，同事们却昏昏欲睡。这件事之后，公司的技术交流计划也就不了了之了。可是我并不想让自己精心准备的培训计划就此流产。既然他们暂时不愿听，那我干脆写到博客吧。这样既可以帮助更多的新手入门，也方便自己以后查阅。第一篇博文的内容讲的是开发前调试环境的部署。可能因为大部分人都不知道使用花生壳进行本地调试，所以也使得该博文迅速成为博客园当天被最多推荐的博文，这更让我坚定了把这个系列写下去的决心。后来电子工业出版社的编辑找到我，询问我是否愿意出版这个系列。其实刚开始我是拒绝的，也是抱着怀疑对方身份的态度，之后经过一系列的考虑与沟通，最终做了出版的决定。“duang”了半年多，书



稿终于完成。

写给谁看

本书所有的代码都是用 C#编写的，所以本书非常适合具有 C#基础的开发人员参阅。在写作的过程中笔者也是比较纠结的，总力求让所有开发人员都能很轻松地看懂。但由于自身技术条件、语言组织能力有限以及微信官方个别文档的不完善，因此文中难免会出现错误的地方。恳请各位同人不吝赐教，批评指正。

怎么看

本书共 10 章，是对微信绝大部分接口的解读（部分接口文档不完善），以及笔者结合实际项目的经验总结。

第 1 章简要介绍微信公众平台的三种分类和使用模式，使读者可以对公众平台有一个初步了解。

第 2 章是开发微信公众平台的基础准备以及阅读本书的基础准备，主要描述公众平台的工作原理和开发环境的部署。

第 3 章对公众平台的对话接口进行讲解，包括基础的素材管理、接收用户消息以及回复用户的处理方式。另外还包括高级接口的群发、模板消息通知、带参数二维码、自定义菜单等接口的使用。

第 4 章和第 5 章是对公众平台用户管理接口和客服管理接口的讲解。由于公众号开发是以用户为基础的，而完善的客服系统对提高客户体验也是至关重要的，因此这两章是全书的重点章节。

第 6 章主要讲的是 JS-SDK 相关接口。JS-SDK 接口使开发者在开发微信平台关联的 Web 应用时有更好的扩展。分享接口为微信的朋友圈营销提供了良好的技术支持。

第 7 章和第 8 章主要讲的是微信支付和微信小店。微信支付是打造微信 O2O 生态圈的重要一环，也是开发者在开发过程中最容易出错的地方。而微信小店是微信电商的一个平台支持。这两章对微信支付和微信小店的各个接口进行详细的讲解。

第 9 章主要讲的是微信卡券的使用。微信卡券的出现使商户进行微信营销时多了一种营销的手段。本章从基础的创建卡券、投放卡券、核销卡券，再到飞机票、电影票、会员卡等特殊卡券的使用进行详细的讲解。



第10章主要讲解笔者在实际项目中遇到的一些比较经典的功能案例。本章的4个案例涵盖了全书大部分的接口使用，力求读者能对各个接口的使用场景与调用方式进行充分的了解。

致谢

感谢博文视点编辑陈晓猛对于本书写作时的帮助。

感谢球哥与飞哥在我写作过程中的支持与帮助。

感谢我最亲爱的家人以及亲爱的她对于我的支持与付出。

最后，谨以此书献给所有致力于微信公众平台开发的朋友们，愿我们在追求技术成就的道路上一起成长。

为便于读者快速上手，本书配套光盘中还有辅助学习视频内容，包括工作原理与调试环境部署、新手接入指南、接收消息、HTTP 请求的代码封装、微信公众平台基础服务、素材管理、被动响应消息与消息体加解密、发送客服消息、高级群发、模板消息、带参数二维码、自定义菜单、订阅用户管理、多客服、JS-SDK、微信支付、微信小店、微信卡券等。另外，本书的相关源代码等资源可以通过 <http://www.broadview.com.cn/27116> 下载。由于微信官方经常会对接口进行调整，因此为了大家都能及时获取到最新的代码，本书的所有源代码还托管到了淘宝开放平台（<http://code.taobao.org>，读者进入该平台后，搜索“lpwxsdk”即可找到最新源代码）；或者直接使用 svn 地址（<http://code.taobao.org/svn/lpwxsdk/>）进行更新，作者会定期修改代码中的 bug，以及更新微信官方的接口调整。

张思凯

2015年8月



目录



第1章 微信公众平台初探 / 1

1.1 微信公众平台分类 / 1

1.1.1 订阅号 / 2

1.1.2 服务号 / 2

1.1.3 企业号 / 3

1.2 公众平台的两种使用模式 / 5

1.2.1 编辑模式 / 5

1.2.2 开发者模式 / 6

第2章 公众号原理与开发环境部署 / 8

2.1 工作原理 / 8

2.1.1 HTTP 请求与响应 / 8

2.1.2 XML 与 JSON 的序列化和反序列化 / 18

2.1.3 官方调试工具使用 / 21

2.2 开发环境的部署 / 23

2.2.1 IIS 安装与部署 / 23

2.2.2 花生壳域名映射与 ngrok / 25

2.2.3 Visual Studio 本地调试 / 29

2.2.4 新手接入指南 / 32

第3章 微信对话服务 / 37

3.1 基础支持 / 37

3.1.1 全局返回码与接口频率限制 / 37



- 3.1.2 获取 access_token / 43
- 3.1.3 获取微信服务器 IP 地址 / 48
- 3.2 素材管理接口 / 50
 - 3.2.1 新增素材 / 50
 - 3.2.2 根据 media_id 获取临时素材 / 57
 - 3.2.3 根据 media_id 获取永久素材 / 59
 - 3.2.4 删除永久素材 / 62
 - 3.2.5 修改永久图文素材 / 63
 - 3.2.6 获取永久素材总数 / 64
 - 3.2.7 获取永久素材列表 / 66
- 3.3 接收消息 / 69
 - 3.3.1 普通消息实体映射 / 70
 - 3.3.2 事件消息实体映射 / 80
 - 3.3.3 消息数据包解析 / 86
 - 3.3.4 消息处理页面示例 / 93
- 3.4 被动响应消息与客服接口 / 99
 - 3.4.1 被动响应消息 / 99
 - 3.4.2 客服接口 / 108
- 3.5 高级群发接口 / 117
 - 3.5.1 上传图文消息素材 / 117
 - 3.5.2 根据分组进行群发 / 119
 - 3.5.3 根据 openid 列表群发和预览消息 / 127
 - 3.5.4 事件推送群发结果 / 130
 - 3.5.5 查询群发消息发送状态 / 134
 - 3.5.6 删除群发 / 135
- 3.6 业务通知模板消息 / 136
 - 3.6.1 设置公众号所属行业 / 136
 - 3.6.2 获取模板 ID / 140
 - 3.6.3 发送模板消息 / 143
 - 3.6.4 模板消息事件推送 / 146
- 3.7 推广支持 / 147



- 3.7.1 生成带参数的二维码 / 147
- 3.7.2 扫描带参数二维码事件处理 / 153
- 3.7.3 长短链接转换接口 / 154
- 3.8 自定义菜单 / 155
 - 3.8.1 自定义菜单创建接口 / 156
 - 3.8.2 自定义菜单查询接口 / 162
 - 3.8.3 自定义菜单删除接口 / 163
 - 3.8.4 自定义菜单事件推送 / 163
- 3.9 消息体签名及加解密 / 172
 - 3.9.1 加解密模式介绍 / 173
 - 3.9.2 接入指南 / 173

第4章 订阅用户管理 / 179

- 4.1 分组管理接口 / 179
 - 4.1.1 创建分组 / 179
 - 4.1.2 查询所有分组 / 181
 - 4.1.3 查询用户所在的分组 / 183
 - 4.1.4 修改分组名 / 184
 - 4.1.5 移动用户分组 / 184
- 4.2 用户信息管理 / 186
 - 4.2.1 获取用户基本信息 / 186
 - 4.2.2 设置用户备注名 / 189
 - 4.2.3 获取用户列表 / 189
- 4.3 OAuth 网页授权获取用户基本信息 / 191
 - 4.3.1 网页授权回调域名设置 / 191
 - 4.3.2 同意授权, 获取 code / 194
 - 4.3.3 通过 code 换取网页授权 access_token / 197
 - 4.3.4 刷新 access_token / 199
 - 4.3.5 拉取用户信息 / 200



第5章 多客服接口 / 201

- 5.1 多客服简介与开通 / 201
- 5.2 将消息转发到多客服 / 202
- 5.3 客服管理 / 205
 - 5.3.1 设置客服账号 / 205
 - 5.3.2 上传客服头像 / 207
 - 5.3.3 删除客服账号 / 208
 - 5.3.4 获取在线客服接待信息 / 208
 - 5.3.5 获取客服基本信息 / 210
 - 5.3.6 获取客服聊天记录接口 / 212
- 5.4 多客服会话控制 / 216
 - 5.4.1 会话状态通知事件 / 216
 - 5.4.2 会话创建与关闭 / 219
 - 5.4.3 获取客户的会话状态 / 221
 - 5.4.4 获取客服的会话列表 / 222
 - 5.4.5 获取未接入会话列表 / 224
- 5.5 PC 客户端自定义插件接口 / 225
 - 5.5.1 接口调试 / 226
 - 5.5.2 向会话窗口输入框中输入一条消息 / 228
 - 5.5.3 高亮自定义插件 Tab 页 / 232
 - 5.5.4 事件接口 / 232

第6章 微信 JS-SDK / 235

- 6.1 JS-SDK 使用步骤 / 235
- 6.2 分享接口 / 244
- 6.3 图像接口 / 247
- 6.4 音频接口 / 248
- 6.5 地理位置 / 252
- 6.6 界面操作 / 255
- 6.7 微信扫一扫接口 / 258





6.8 其他 JS 接口 / 259

第 7 章 支付接口开发 / 260

7.1 微信支付简介 / 260

7.2 接口调用规则 / 261

7.2.1 协议规则 / 261

7.2.2 参数规定 / 261

7.2.3 安全规范 / 262

7.3 统一下单接口 / 265

7.4 支付结果通用通知 / 274

7.5 查询订单接口 / 283

7.6 JSAPI (网页内) 支付接口 / 285

7.6.1 场景交互细节 / 285

7.6.2 获取当前微信版本号 / 287

7.6.3 显示微信安全支付标题 / 287

7.6.4 JavaScript 调用支付 API / 288

7.6.5 网页内支付示例 / 289

7.7 扫码支付 / 297

7.7.1 扫描支付——模式一 / 297

7.7.2 扫描支付——模式二 / 304

7.8 刷卡支付 / 304

7.9 撤销订单 / 308

7.10 关闭订单 / 310

7.11 退款 API / 312

7.12 商户营销与支付工具 / 323

7.12.1 代金券或立减优惠 / 323

7.12.2 现金红包 / 327

7.12.3 企业付款 / 332

第 8 章 微信小店开发 / 337

8.1 微信小店的开通与搭建 / 337

8.2 商品管理 / 339



- 8.2.1 获取指定分类的所有子分类 / 342
- 8.2.2 获取指定子分类的所有 SKU / 344
- 8.2.3 获取指定分类的所有属性 / 346
- 8.2.4 增加商品 / 348
- 8.2.5 修改商品 / 357
- 8.2.6 查询商品 / 358
- 8.2.7 删除商品 / 360
- 8.2.8 商品上下架 / 361
- 8.2.9 修改商品库存 / 362
- 8.3 邮费模板管理接口 / 363
 - 8.3.1 增加邮费模板 / 363
 - 8.3.2 删除邮费模板 / 367
 - 8.3.3 修改邮费模板 / 368
 - 8.3.4 获取邮费模板 / 369
- 8.4 商品分组管理 / 371
 - 8.4.1 增加分组 / 371
 - 8.4.2 删除分组 / 372
 - 8.4.3 修改分组名 / 373
 - 8.4.4 修改分组商品 / 373
 - 8.4.5 获取分组信息 / 375
- 8.5 货架管理 / 378
 - 8.5.1 增加货架 / 378
 - 8.5.2 删除货架 / 388
 - 8.5.3 修改货架 / 389
 - 8.5.4 获取货架信息 / 389
 - 8.5.5 自定义货架页面 / 393
- 8.6 订单管理 / 394
 - 8.6.1 订单付款通知 / 394
 - 8.6.2 根据订单 ID 获取订单详情 / 395
 - 8.6.3 根据订单状态/创建时间获取订单列表 / 400
 - 8.6.4 设置订单发货信息 / 401



8.6.5 关闭订单 / 404

第9章 卡券功能接口 / 406

9.1 功能简介 / 406

9.2 开发流程 / 406

9.3 创建卡券前的准备 / 407

9.3.1 上传 LOGO 接口 / 407

9.3.2 门店管理接口 / 408

9.3.3 获取颜色列表接口 / 416

9.4 CreateCard 创建卡券接口 / 418

9.5 卡券投放 / 438

9.5.1 创建二维码 / 438

9.5.2 获取 api_ticket / 440

9.5.3 批量添加卡券接口 / 442

9.6 卡券核销 / 444

9.6.1 消耗 code / 444

9.6.2 调起卡券列表并获取用户选择列表 / 446

9.7 卡券管理 / 448

9.7.1 删除卡券 / 448

9.7.2 查询 code / 449

9.7.3 批量查询卡列表 / 451

9.7.4 查询卡券详情 / 452

9.7.5 事件推送 / 456

9.7.6 更改 code / 459

9.7.7 设置卡券失效接口 / 460

9.7.8 更改卡券信息接口 / 460

9.7.9 库存修改接口 / 465

9.8 特殊卡票操作 / 466

9.8.1 会员卡 / 466

9.8.2 电影票 / 470

9.8.3 飞机票在线值机 / 471



9.8.4 更新会议门票 / 472

9.9 设置测试白名单 / 474

第 10 章 应用案例 / 476

10.1 微信扫一扫登录 PC 网站 / 476

10.2 网页分享——我是人气王 / 481

10.3 共享用户收货地址 / 489

10.4 微信卡券应用——电影票 / 492

第1章 微信公众平台初探

21 世纪是互联网的时代，各行各业都在和互联网产生着或多或少的关系，因此也产生了很多互联网巨头，国外的微软、谷歌、Facebook、推特，国内的阿里、百度与腾讯。随着智能手机的发展，移动互联网也在近几年迅速发展，4G 时代的开启以及移动终端的凸显必将为移动互联网的发展注入巨大的能量。对于像笔者这样的“屌丝”开发者来说，是一个很好的契机，正如“乱世出英雄”，而如今的这个年代，互联网带给我们的机会也是很多的。各大互联网巨头也都在搭建自己的生态圈，以此来吸引开发者，从而吸引更多的用户。而微信公众平台就是在这种环境下产生的。

1.1 微信公众平台分类

微信（WeChat）是腾讯公司于 2011 年 1 月 21 日推出的一个为智能手机提供即时通信服务的 App。其支持跨通信运营商、跨操作系统平台，仅需消耗少量网络流量，即可通过网络快速发送免费的语音短信、视频、图片和文字，同时也具有丰富的内置游戏，以及社交类插件“摇一摇”、“漂流瓶”和“朋友圈”。截至 2013 年 11 月，其注册用户量已经突破 6 亿，是亚洲地区最大用户群体的移动即时通信软件。其实在微信推出之前一个月的时候，小米公司就已经推出和微信功能相似的米聊了；而微信现阶段的发展，米聊却无法望其项背。微信之所以能取得今日的成绩，和腾讯在即时通信的霸主地位是分不开的。然而，真正让微信成为现阶段大众日常所用的工具的大功臣，正是本书要讲的微信公众平台。



微信公众平台是继微博之后唯一一个被大众接受并热烈追捧的营销产品。和新浪微博早期从明星战略着手不同,微信拥有亿级用户,挖掘用户的价值,为这个平台增加更优质的内容,创造更好的黏性,形成一个闭合的生态循环,是平台发展初期更重要的方向。以社交和分享为基础的微信,天生就适合进行自媒体营销活动。为了适合不同的使用场景,微信公众平台在2013年8月将公众平台分为订阅号和服务号。2013年10月底,开放了全新的认证体系,支持服务号进行微信认证,通过微信认证了的服务号可获得认证标识,并同时开通高级接口权限。可开通的接口包括语音识别、客服接口、OAuth2.0网页授权、生成带参数的二维码、获取用户地理位置、获取用户基本信息、获取关注者列表、用户分组接口、上传下载多媒体文件。腾讯官方也许是发现在公众平台中,有越来越多的企业在建立自己企业内部的公众号,从而将公众平台打造成移动端的OA管理系统,于是在2014年9月,该公司推出公众企业号,定位于企业内部管理,可无缝和OA对接。

1.1.1 订阅号

订阅号主要是为媒体和个人提供一种新的信息传播方式,构建与读者之间更好的沟通与管理模式,订阅号的侧重点是信息的传播与分享。在最初刚刚推出订阅号时,订阅号只有简单的消息的接收与回复,以及群发功能。虽然订阅号每天有一次的群发机会,但由于订阅号的群发会折叠在用户微信的聊天列表的“订阅号”文件夹中,如图1-1所示,因此订阅用户只有在打开“订阅号”文件夹后,才能看到折叠在里面的订阅号发送的消息。用户无法第一时间发现并阅读公众号发送过来的消息,消息的点击率偏低。

随着微信的发展,官方逐步开放了更多的接口,但前提是必须通过微信认证。如自定义菜单接口、高级群发接口、卡券功能、用户管理接口等。腾讯的开放性也在慢慢地模糊微信公众号与App的界限。

1.1.2 服务号

服务号是给企业和组织提供更强大的业务服务与用户管理能力,帮助企业快速实现全新的公众号服务平台,打造真正意义上的“App”。服务号的侧重点是用户管理与营销推广,可利用的接口也比订阅号灵活多样。1.1.1节说到订阅号每天的一次群发机会,服务号的群发机会只有每月四次。虽然相对于订阅号来说,服务号群发的机会少了很多;但服务号发送的消息则是像用户的好友发送的消息一样,显示在好友的对话列表中,被用户查看的可能性大了很多,也就更利于传播。



图 1-1

1.1.3 企业号

企业号为企业或组织提供移动应用入口,帮助企业建立与员工、上下游供应链及企业应用间的连接。企业号可以理解为企业的移动 OA 管理系统,微信提供给企业号的接口也比较多样化。企业号具备如下特点。

- 关注更安全。企业管理员事先将成员导入通讯录,对方需验证身份才能关注企业号。
- 应用可配置。每个企业号可自由配置多个“子号”,来对接企业的多种不同应用,每个子号的功能相当于一个独立的服务号。



- 消息无限制。可自由推送消息，并使用微信的原生能力，满足企业的各种应用场景。
- 使用更便捷。企业号出现在微信会话列表首层，在通讯录中有单独的分类。

总而言之，不管是订阅号、服务号，还是企业号，微信的开放姿态是有目共睹的。微信公众平台的发展给了市场很大的机会，也给了开发者很大的机会，抓住了这个机会，我们可能在以后的行业竞争中，就多了点本钱，多了点竞争力。开发者在开发微信公众号之前，需要先认真了解各个类型的公众号的区别以及不同类型的公众号所具备的功能权限。表 1-1 所示的就是各个类型公众号之间的区别。

表 1-1

账号类型	订 阅 号		服 务 号		企 业 号	
业务介绍	为媒体和个人提供一种新的信息传播方式，构建与读者之间更好的沟通与管理模式		给企业和组织提供更强大的服务与用户管理能力，帮助企业实现全新的公众号服务平台		帮助企业和组织内部建立员工、上下游合作伙伴与企业IT系统间的连接	
适用人群	个人和组织		不适用于个人		企业、政府、事业单位或其他组织	
功能权限	普通订阅号	微信认证订阅号	普通服务号	微信认证服务号	普通企业号	微信认证企业号
消息直接显示在好友对话框列表中			✓	✓	✓	✓
消息显示在“订阅号”文件夹中	✓	✓				
每天可以群发一条消息	✓	✓				
每个月可以群发四条消息			✓	✓		
无限制群发					✓	✓
保密消息禁止转发					✓	✓
关注时验证身份					✓	✓
基本的消息接收/回复接口	✓	✓	✓	✓	✓	✓
自定义菜单	✓	✓	✓	✓	✓	✓
定制应用					✓	✓
高级接口能力		部分支持		✓		部分支持
微信支付-商户功能				✓		



1.2 公众平台的两种使用模式

为了满足大部分用户的需求，微信官方提供了两种使用微信公众号的方式。其中一种主要是面向无开发能力，并且也没有太多定制需求的用户；另一种就是具备开发能力或者有条件进行定制开发的用户。由于第一种模式不是本书的重点，所以在此简要介绍一下，读者了解一下就行了，没必要深究。

1.2.1 编辑模式

进入微信公众平台后台 (<https://mp.weixin.qq.com/>)，在左边导航栏中，进入“功能”→“自动回复”，如图 1-2 所示。



图 1-2

可以看到，这里有三种回复类型，分别是被添加自动回复、消息自动回复、关键词



自动回复。下面的富文本框是定义回复内容的。被添加自动回复指的是当用户关注公众号时回复给用户的内容，可设置文字/语言/图片/视频为被添加自动回复内容，如图 1-3 所示。

微信中的效果如图 1-4 所示。

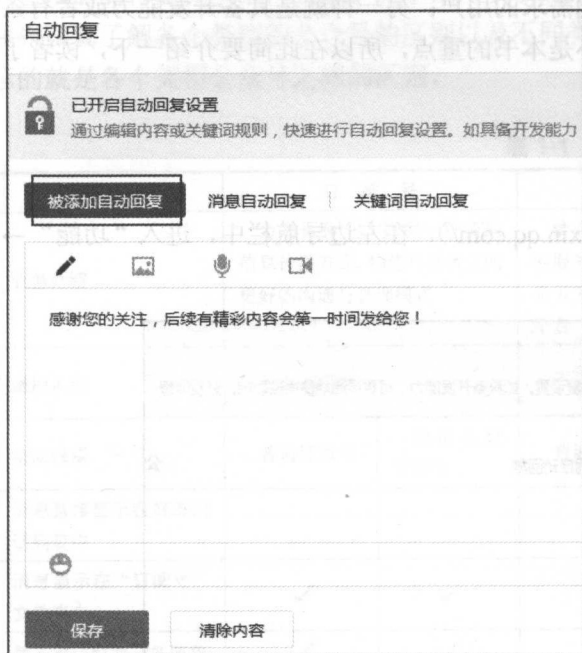


图 1-3

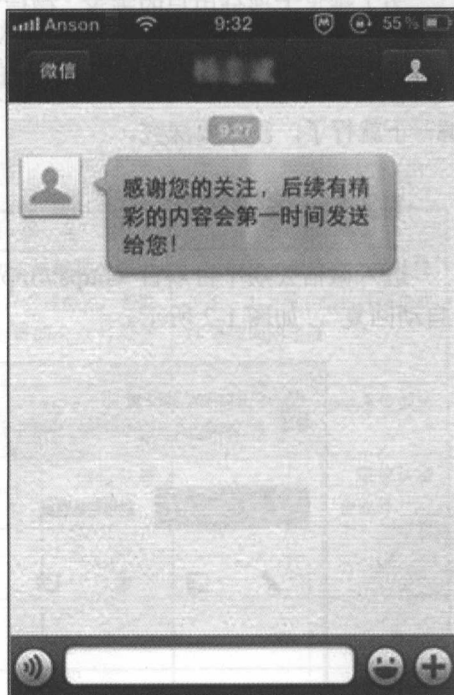


图 1-4

消息自动回复和关键词自动回复以及消息群发的操作都和上面的实例类似，在此不再赘述，本书的重点是开发者模式中的开发接口。

1.2.2 开发者模式

登录微信公众平台后，在左边导航栏中，进入“开发者中心”，在“开发者中心”可以看到 AppID 和 AppSecret，以及服务器配置相关项，在此页面的最下面是接口权限表（见图 1-5 和图 1-6）。关于服务器配置相关项，将在以后的章节详细讲述。



配置项	接口报警
<p>开发者ID</p> <p>AppID(应用ID) wx8ef...:34114</p> <p>AppSecret(应用密钥) 80*****f9 完整显示 重置</p>	
<p>服务器配置(已启用) 修改</p> <p>停用服务器配置后, 用户消息和开发者需要的事件推送, 将不会被转发到该URL中</p> <p>URL(服务器地址) http://te...l.com/wx.ashx</p> <p>Token(令牌) ...</p> <p>EncodingAESKey(消息加解密密钥) dD26XtlxqJrURMDC3pmTTK9tqjGE2vx3oLq3ohxofL</p> <p>消息加解密方式 安全模式</p>	

图 1-5

类目	功能	接口	每日调用上限/次	接口状态
基础支持		获取access_token	2000	已获得
		获取微信服务器IP地址		已获得
接收消息	接收消息	验证消息真实性	无上限	已获得
		接收普通消息	无上限	已获得
		接收事件推送	无上限	已获得
		接收语音识别结果 (已开启)	无上限	已获得
		自动回复	无上限	已获得
发送消息	发送消息	客服接口		未获得
		群发接口		未获得
		模板消息 (业务通知)		未获得
对话服务	对话服务	用户分组管理		未获得
		设置用户备注名		未获得

图 1-6

第2章 公众号原理与开发环境部署

俗话说“工欲善其事，必先利其器”，也就是说，要做好一件事，准备工作非常重要，准备工作做得充足，才能在做事的过程中得心应手。使用微信公众平台接口做开发前，首先要了解微信公众平台的工作原理，其次是部署应有的开发环境。本书中的所有代码都是基于C#语言进行开发的，所以，IIS、Visual Studio 是必不可少的，还要有相应的开发调试工具等。

2.1 工作原理

直观上，我们看到的微信公众号的工作流程是这样的：用户发送消息或者单击公众号的自定义菜单，服务器收到用户的请求后，根据业务逻辑回复给用户对应的消息，如文本、图文、视频、音频等。学过网络的人看着一定很熟悉，这不就是典型的 HTTP 请求嘛。

2.1.1 HTTP 请求与响应

HTTP 协议全称为超文本传输协议，定义了浏览器怎样向万维网（也可以叫网站或网络程序）发送请求，以及万维网服务器怎样响应用户的请求。每个万维网都有一个服务器进程监听这个万维网设置的 TCP 端口，默认为 80。一旦发现浏览器向它发出连接建立请求，则建立 TCP 连接，服务器接收到用户的请求后，根据用户的请求，结合实际的业务逻辑，返回所请求的页面作为响应。最后，TCP 连接被释放，如图 2-1 所示。

HTTP 协议的主要特点概况如下。

- 支持客户端/服务器模式。

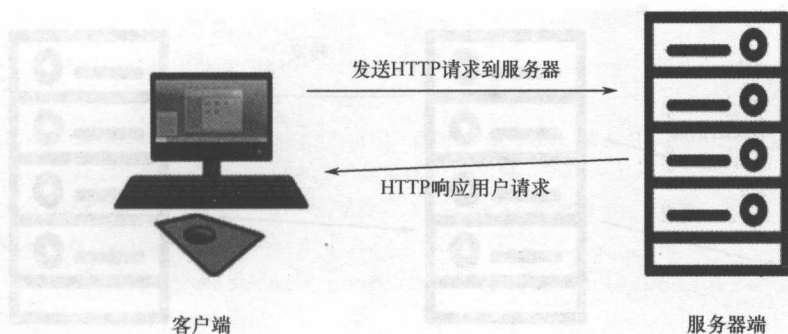


图 2-1

- 简单快速。客户端向服务器端请求服务时，只需传送请求方法和路径，请求方法常用的有 GET、POST。每种方法规定了客户与服务器联系的类型不同。
- 无状态。HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大，如 ASP.NET 中的 VIEWSTATE。

HTTP 协议常用请求头如下。

- Accept: 浏览器可接受的 MIME 类型，指的是浏览器支持的 MIME 类型。常用的有 ext/html、application/xhtml+xml、application/xml 、 */* 等。
- ContentType: 内容类型，指网页中存在的 Content-Type，用户定义网络文件的类型和网页的编码，决定浏览器将以什么样的形式、什么编码读取这个文件。指定响应的 HTTP 内容类型，如果指定为 ContentType，则默认为 TEXT/HTML。

微信公众平台的自动回复功能就和 HTTP 的原理一致，而开发者根据微信官方提供的接口来定制自己的公众平台时，就稍微有点差别了。首先，当用户在微信里给公众号发送消息或者单击微信自定义菜单时，微信手机客户端将这条消息发送到微信服务器。其次，微信服务器收到用户的请求后，根据具体的业务逻辑再将消息推送给开发者设置的 URL，这个 URL 对于微信服务器来说就是一个服务器。微信服务器此时充当的是客户端的角色。开发者的服务器收到微信服务器推送过来的请求后，根据微信提供的规则解析出用户发送的消息内容，然后再根据具体的业务逻辑将响应的内容回复给微信服务器，微信服务器再返回给用户。在整个过程中，开发者的服务器对于用户来说是透明的，如图 2-2 所示。

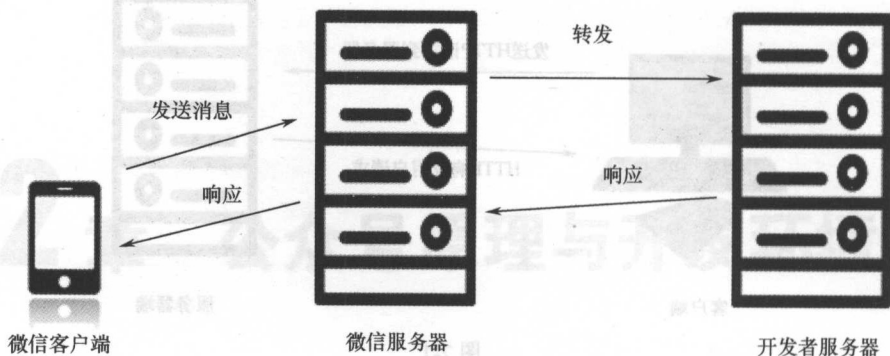


图 2-2

了解了 HTTP 的基本原理后,就需要根据 HTTP 协议写出可以与微信服务器通信的 C# 代码了。

首先,定义一个类 Utils,类中定义一个方法 HttpGet,此方法是用来发送 HTTP 的 GET 请求的。在该方法中,使用 System.Net 命名空间下的 WebRequest 类的 Create 方法,初始化一个请求实例,如代码清单 2-1 所示。

代码清单 2-1

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
```

然后,设置常用的请求头,如代码清单 2-2 所示。

代码清单 2-2

```
request.Method = "GET"; // 设置请求的方法
request.Accept = "*/*"; // 设置 Accept 标头的值
```

最后,调用请求实例的 GetResponse()方法,获取响应流,并将响应流转换成字符串。完整代码如代码清单 2-3 所示。

代码清单 2-3

```
/// <summary>
/// HTTP GET 方式请求数据
/// </summary>
/// <param name="url">请求的 url</param>
```



```

/// <returns>响应信息</returns>
public static string HttpGet(string url)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
    request.Method = "GET"; //设置请求的方法
    request.Accept = "*/*"; //设置 Accept 标头的值
    string responseStr = "";
    using (HttpWebResponse response = (HttpWebResponse)request.
        GetResponse()) //获取响应
    {
        using (StreamReader reader =
            new StreamReader(response.GetResponseStream(), Encoding.UTF8))
        {
            responseStr = reader.ReadToEnd();
        }
    }
    return responseStr;
}

```

HttpGet 方法的功能是发送 GET 请求。我们在浏览器中浏览某一个网页时，发送的也是一个 GET 请求。如图 2-3 所示，访问的网络是百度，在浏览器的开发者工具中可以看到几个 Tab。其中在 Headers 中，我们可以看到 Request Method 的值为 GET，说明这个请求是 GET 请求，在 Response 中显示的是这个请求的响应。

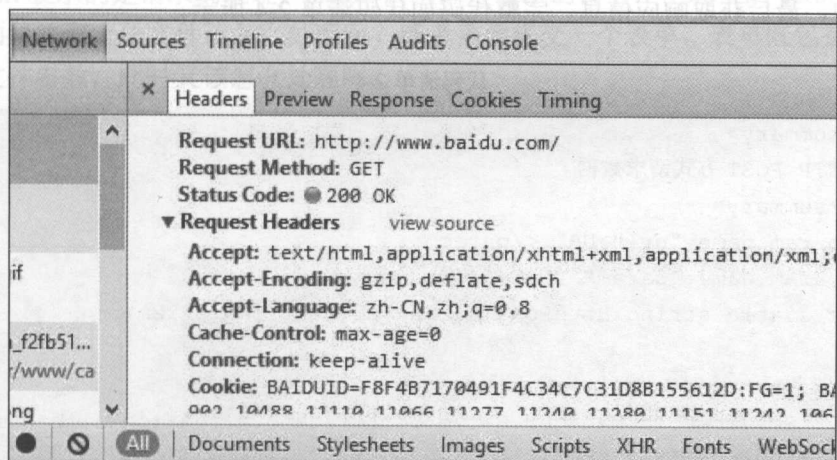


图 2-3



下面测试一下 `HttpGet` 方法的运行结果，一个很简单的测试如下：请求百度的官方网址，然后将响应的结果输出到页面上。代码如下：

```
Response.Write(Utils.HttpGet("http://www.baidu.com"));
```

执行结果如图 2-4 所示。我们会发现和直接在浏览器中输入 `http://www.baidu.com` 一样。



图 2-4

同 `GET` 请求一样，`POST` 请求也需要先创建一个连接实例。不同的是我们需要设置 `Method` 为 `POST`，`ContentType` 为 `application/x-www-form-urlencoded`，然后将请求的数据写入请求流中，最后获取响应信息。完整代码如代码清单 2-4 所示。

代码清单 2-4

```
/// <summary>
/// HTTP POST 方式请求数据
/// </summary>
/// <param name="url">URL.</param>
/// <param name="param">POST 的数据</param>
public static string HttpPost(string url, string param)
{
    //当请求为 https 时，验证服务器证书
    ServicePointManager.ServerCertificateValidationCallback=new
    RemoteCertificateValidationCallback((a,b,c,d) => true );
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
```



```
request.Method = "POST";
request.ContentType = "application/x-www-form-urlencoded";
request.Accept = "*/*";
request.Timeout = 15000;
request.AllowAutoRedirect = false;
string responseStr = "";
using (StreamWriter requestStream =
new StreamWriter(request.GetRequestStream()))
{
    requestStream.Write(param); //将请求的数据写入请求流
}
using (HttpWebResponse response = (HttpWebResponse) request.GetResponse())
{
    using (StreamReader reader
=new StreamReader(response.GetResponseStream(), Encoding.UTF8))
    {
        responseStr = reader.ReadToEnd(); //获取响应
    }
}
return responseStr;
}
```

上述代码中的方法 `HttpPost` 接收两个参数：一个 `string` 类型的 `url`，一个 `string` 类型的 `param`。`url` 表示的是请求的地址，`param` 表示的是请求的内容，但这里的 `param` 是 `string` 类型的。如果需要上传文件该怎么处理呢？或者需要提交一个表单，表单既包括文本参数，又包括文件参数，此时又该怎么处理呢？

通常情况下，POST 提交带文件的表单类似于代码清单 2-5 所示。

代码清单 2-5

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.
cs" Inherits="WxTest.WebForm1" %>
<!DOCTYPE html>
<html>
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<script type="text/C#" runat="server">
    protected override void OnLoad(EventArgs e)
```




```
{
    if (!string.IsNullOrEmpty(Request.Form["t1"]))
    {
        Response.Write("2");
        Response.End();
    }
}
</script>
</head>
<body>
    <form id="form1" method="POST" enctype="multipart/form-data">
        <input type="text" name="t1"/>
        <input type="text" name="t2"/>
        <input type="text" name="t3"/>
        <input type="file" name="t4"/>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

在上述代码中，表单有三个文本框、一个 file 控件和一个提交按钮。在单击“提交”按钮前，按“F12”键，打开开发者工具，并切换到“网络”Tab，如图 2-5 所示。

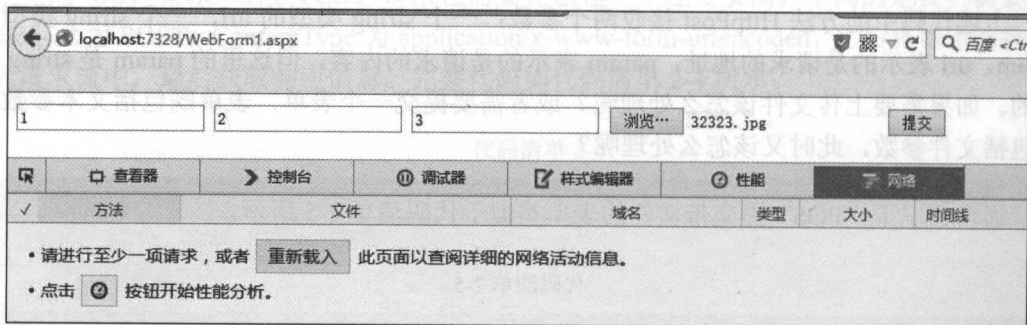


图 2-5

输入表单后，单击“提交”按钮，此时应该可以看到一个 POST 请求的记录。单击此条请求，在右边栏中切换到“参数”Tab，如图 2-6 所示。

从图 2-6 中可以发现，每个参数都很规律：第一行由几个下画线和一个随机字符串组成，第二行则是和表单相关的信息，然后空一行，第四行则是表单的值。文件的信息块和



文本的信息块多了一行 Content-Type:image/jpeg，文件信息块的最下面为文件的内容。最后再由一个分割字符串作为表单的结束标记（限于篇幅，结束标记未截出，有兴趣的读者可自行实验）。

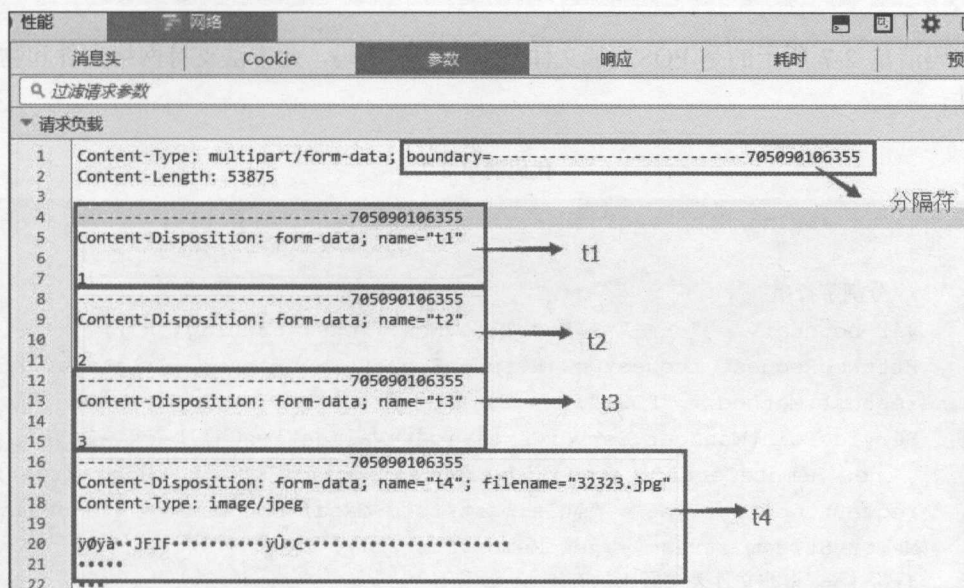


图 2-6

知道了规则，我们只需按照这个规则拼接请求的数据包即可。首先，为了方便调用，将表单中的项封装为实体类，如代码清单 2-6 所示。

代码清单 2-6

```
public class FormEntity
{
    /// <summary>
    /// 表单名
    /// </summary>
    public string Name { get; set; }
    /// <summary>
    /// 表单值
    /// </summary>
    public string Value { get; set; }
    /// <summary>
```



```
/// 是否是文件  
/// </summary>  
public bool IsFile { get; set; }  
}
```

代码清单 2-7 所示的是 POST 带文件表单的实现代码，此方法支持网络文件和物理路径文件。

代码清单 2-7

```
public static string HttpPostForm(string url, List<FormEntity> form)  
{  
    //分割字符串  
    var boundary = "----" + DateTime.Now.Ticks.ToString("x");  
    HttpRequest request = (HttpRequest)WebRequest.Create(url);  
    request.Method = "POST";  
    ServicePointManager.ServerCertificateValidationCallback =  
        new RemoteCertificateValidationCallback((a, b, c, d) => true);  
    request.ContentType = "multipart/form-data; boundary=" + boundary;  
    MemoryStream stream = new MemoryStream();  
    #region 将非文件表单写入内存流  
    foreach (var entity in form.Where(f => f.IsFile == false))  
    {  
        var temp = new StringBuilder();  
        temp.AppendFormat("\r\n--{0}", boundary);  
        temp.Append("\r\nContent-Disposition: form-data;");  
        temp.AppendFormat("name=\"{0}\"", entity.Name);  
        temp.Append("\r\n\r\n");  
        temp.Append(entity.Value);  
        byte[] b = Encoding.UTF8.GetBytes(temp.ToString());  
        stream.Write(b, 0, b.Length);  
    }  
    #endregion  
    #region 将文件表单写入内存流  
    foreach (var entity in form.Where(f => f.IsFile == true))  
    {  
        byte[] filedata = null;  
        var filename = Path.GetFileName(entity.Value);  
        //表示网络资源
```



```
if (entity.Value.Contains("http"))
{
    //处理网络文件
    using (var client = new WebClient())
    {
        filedata = client.DownloadData(entity.Value);
    }
}
else
{
    //处理物理路径文件
    using (FileStream file = new FileStream(entity.Value,
        FileMode.Open, FileAccess.Read))
    {
        filedata = new byte[file.Length];
        file.Read(filedata, 0, (int)file.Length);
    }
}

var temp = string.Format("\r\n--{0}\r\nContent-Disposition: " +
    "form-data; name=\"{1}\"; filename=\"{2}\" \r\n\r\n",
    boundary, entity.Name, filename);
byte[] b = Encoding.UTF8.GetBytes(temp);
stream.Write(b, 0, b.Length);
stream.Write(filedata, 0, filedata.Length);
}
#endregion
//结束标记
byte[] foot_data = Encoding.UTF8.GetBytes("\r\n--" + boundary +
    "--\r\n");
stream.Write(foot_data, 0, foot_data.Length);
Stream reqStream = request.GetRequestStream();
stream.Position = 0L;
//将 Form 表单生成流写入请求流
stream.CopyTo(reqStream);
stream.Close();
reqStream.Close();
using (HttpWebResponse response = (HttpWebResponse) request.GetResponse())
{

```




```
using (StreamReader reader =  
new StreamReader(response.GetResponseStream(), Encoding.UTF8))  
{  
    return reader.ReadToEnd(); //获取响应  
}  
}
```

2.1.2 XML 与 JSON 的序列化和反序列化

XML 指的是可扩展标记语言，它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。它非常适合互联网间的数据传输，提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。之所以在这里讲 XML，是因为在微信服务器与开发者服务器间的通信都是以 XML 格式进行数据传输的。当然本书并不会在这里讲 XML 的格式、语法等，我们只需要知道怎么用 C# 来处理 XML 就行了。C# 中 System.Xml.Linq 命名空间下的 XElement 封装了一系列处理 XML 的方法。其中，静态方法 Parse(string text) 的功能是从包含 XML 的字符串中加载 System.Xml.Linq.XElement。为了更便于读者理解，这里用一个例子来说明。代码清单 2-8 是微信服务器推送给开发者服务器的 XML 字符串。

代码清单 2-8

```
<xml>  
  <ToUserName><![CDATA[toUser]]></ToUserName>  
  <FromUserName><![CDATA[fromUser]]></FromUserName>  
  <CreateTime>1348831860</CreateTime>  
  <MsgType><![CDATA[text]]></MsgType>  
  <Content><![CDATA[this is a test]]></Content>  
  <MsgId>1234567890123456</MsgId>  
</xml>
```

当我们接收到微信服务器推送过来的消息时，首先调用 XElement.Parse(string text) 方法，将 XML 字符串转换成 XElement 对象（见代码清单 2-9）。

代码清单 2-9

```
XElement xdoc = XElement.Parse(xml);
```



比如现在想获取 FromUserName 的值，应该这样写（见代码清单 2-10）。

代码清单 2-10

```
string FromUserName = xdoc.Element("FromUserName").Value;
```

执行上面的代码后，FromUserName 的值应该为“fromUser”，其他的字段获取方法同理。

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，它基于 JavaScript 的一个子集。JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯（包括 C、C++、C#、Java、JavaScript、Perl、Python 等）。这些特性使 JSON 成为理想的数据交换语言，易于人阅读和编写，同时也易于机器解析和生成（网络传输速度快）。

JSON 语法规则如下：

- 数据在名称/值对中。
- 数据由逗号分隔。
- 花括号保存对象。
- 方括号保存数组。

和 XML 一样，C#中也有序列化和反序列化 JSON 的类库，Newtonsoft.Json、LitJson 都是比较优秀的.NET 中操作 JSON 的开源类库。这里我们主要讲一下 Newtonsoft.Json。Newtonsoft.Json 的官方地址是：<http://www.newtonsoft.com/products/json/>。下载完成后将压缩包解压，解压后会看到 Bin、Source 两个文件夹。其中 Source 是此类库的源码，有兴趣的读者可以深入研究一下，本书不进行深入讲解。打开 Bin 文件，会看到 Net20、Net35、Net40 等文件夹。不同的文件夹对应不同的 Framework 的版本，使用的时候，选择和自己项目匹配的版本即可。下面的实例讲解了如何使用 Newtonsoft.Json 进行序列化与反序列。

首先，新建一个 WebForm 应用程序，目标框架选.NET Framework 4，在项目中新建一个文件夹“dll”，用来存放外部引用的类库，然后将压缩包中解压出来的 Net40 文件夹中的 Newtonsoft.Json.dll、Newtonsoft.Json.pdb、Newtonsoft.Json.xml 全部复制到 dll 文件中，再将 Newtonsoft.Json.dll 添加到引用即可。最后新建一个测试页面 Test.aspx。在此页面的后台 cs 文件中添加三个方法：ObjToJson、JsonToObj 和 JsonToT。

ObjToJson 代码如代码清单 2-11 所示。



代码清单 2-11

```
void ObjToJson()  
{  
    var obj = new  
    {  
        username="张三",  
        sex="男",  
        age=20  
    };  
    Response.Write(JsonConvert.SerializeObject(obj));  
}
```

此方法首先创建了一个匿名对象，然后调用 Newtonsoft.Json 类库中的 JsonConvert.SerializeObject(Object obj) 方法。此方法的作用是将指定的对象序列化为 JSON 字符串。调用 ObjToJson 方法，执行结果如下：

```
{"username": "张三", "sex": "男", "age": 20}
```

JsonToObj 代码如代码清单 2-12 所示。

代码清单 2-12

```
void JsonToObj(string json)  
{  
    JObject jobj = JsonConvert.DeserializeObject<JObject>(json);  
    Response.Write(jobj.GetValue("username"));  
    Response.Write(jobj.GetValue("sex"));  
}
```

调用 JsonToObj，传入 ObjToJson 执行的结果作为参数，执行的结果是：“张三男”。上述方法中的 JObject 表示 JSON 对象的类，此类继承了 IDictionary<string, JToken>, ICollection<KeyValuePair<string, JToken>>。所以，此类可以和使用 Dictionary 相似的方法进行操作。这个类的使用场景是 JSON 中对象的属性并不多，或者我们只需要其中的几个字段的时候。在转换成对象的时候不需要新建一个类。代码清单 2-13 是 JsonToT 方法的代码。

代码清单 2-13

```
T JsonToT<T>(string json)
```



```
{  
    var t = JsonConvert.DeserializeObject<T>(json);  
    return t;  
}
```

这个方法和 `JsonToObj` 类似。在调用这个方法前，需要根据 JSON 字符串确定对象的属性。比如说要将字符串 `{"username":"张三","sex":"男","age":20}` 转换成对象，首先要定义一个类 `Users`，如代码清单 2-14 所示。

代码清单 2-14

```
public class Users  
{  
    public string username { get; set; }  
    public string sex { get; set; }  
    public int age { get; set; }  
}
```

然后调用方法，如代码清单 2-15 所示。

代码清单 2-15

```
var user = JsonToT<Users>("{\"username\":\"张三\",\"sex\":\"男\",\"age\":20}");
```

`user` 的类型就是 `Users`，对应属性的值为 `user.username="张三"`、`user.sex = "男"`，`user.age=20`。

在微信接口开发的过程中，接收微信服务器推送的消息以及被动回复消息给用户时使用 XML，其他接口调用的时候均使用 JSON。读者需认真掌握 JSON 与 XML 数据的解析，方可快速使用微信提供的接口。

2.1.3 官方调试工具使用

为了方便开发者调试接口，官方提供了调试工具，地址是：<http://mp.weixin.qq.com/debug>。也可以在微信公众平台的后台中，依次选择“开发者中心”→“开发者工具”→“在线接口调试工具”进入，如图 2-7 所示。

进入调试系统后，可以看到系统的使用说明，具体如下。

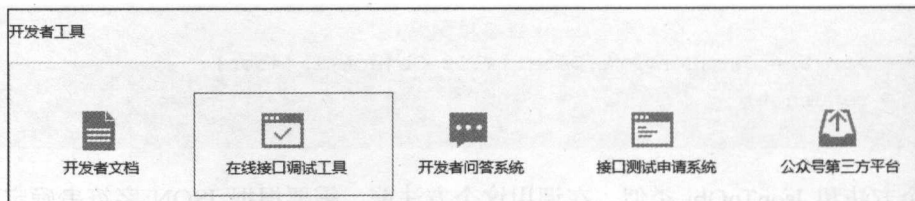


图 2-7

(1) 选择合适的接口。

(2) 系统会生成该接口的参数表，可以直接在文本框内填入对应的参数值（红色星号表示该字段必填）。

(3) 单击“检查问题”按钮，即可得到相应的调试信息。

下面的例子演示的是使用调试系统调用获取 `access_token` 的接口。

第一步，选择接口类型为“基础支持”。

第二步，参数列表选择“获取 `access_token` 接口/token”。

第三步，从“开发者中心”获取 AppID 和 AppSecret，填入对应的文本框，然后单击“检查问题”按钮，执行结果如图 2-8 所示。

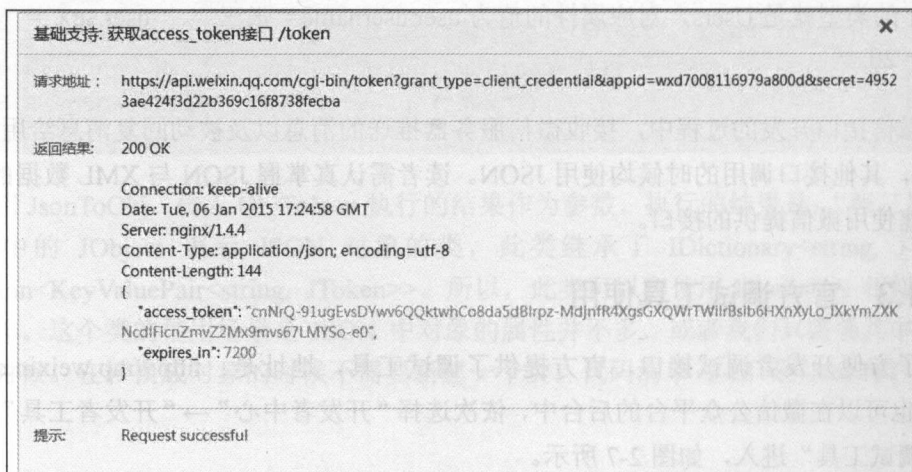


图 2-8



熟练使用调试工具可以方便我们在开发的过程中发现错误。在后面的章节中，本书也会向大家讲述怎样自己动手制作调试工具的案例，敬请期待。

2.2 开发环境的部署

前面讲述了微信公众号的运行原理，以及在开发过程中，数据交换的表示形式。本节进入开发正题，首先需要部署开发与调试环境。由于.NET 开发程序员对 Visual Studio 一定不陌生，本书就不赘述了。但并不是所有的.NET 程序员对 IIS 都比较熟悉，所以在本节中会简要讲解一下 IIS 的安装与部署。从微信的开发原理我们了解到，我们开发的程序是需要作为服务器端程序执行的，正式上线运行的时候可能使用虚拟主机就够了。但在开发的过程中，程序调试就是一个难题。因为微信的服务器需要推送消息给我们的处理程序，这就要求我们在开发的过程中，外网必须能访问到我们的本地计算机（此时对于微信的服务器来讲，我们本地的计算机就是服务器）。也就是说，我们要拥有一台昂贵的服务器。而作为“屌丝”开发者来说，还是能节约点成本就节约点吧，使用动态域名解析工具便能完美解决这个问题。

2.2.1 IIS 安装与部署

Internet Information Services (IIS, 互联网信息服务)是.NET 程序员开发 B/S 架构必需的组件，主要用作 Web 服务器。下面简要讲述 IIS 安装以及安装完成后如何部署 Web 应用程序。依次进入“控制面板”→“程序”→“程序和功能”→“启用或关闭 Windows 功能”，进入“Windows 功能”窗口，如图 2-9 所示。

勾选需要的功能，单击“确定”按钮进行安装。安装成功后，依次单击“开始”→“运行”，输入“inetmgr”，进入 IIS 管理器。图 2-10 所示即 IIS 的管理界面。

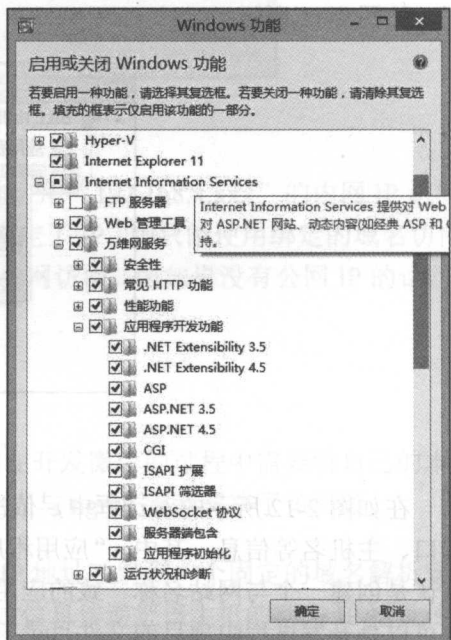


图 2-9

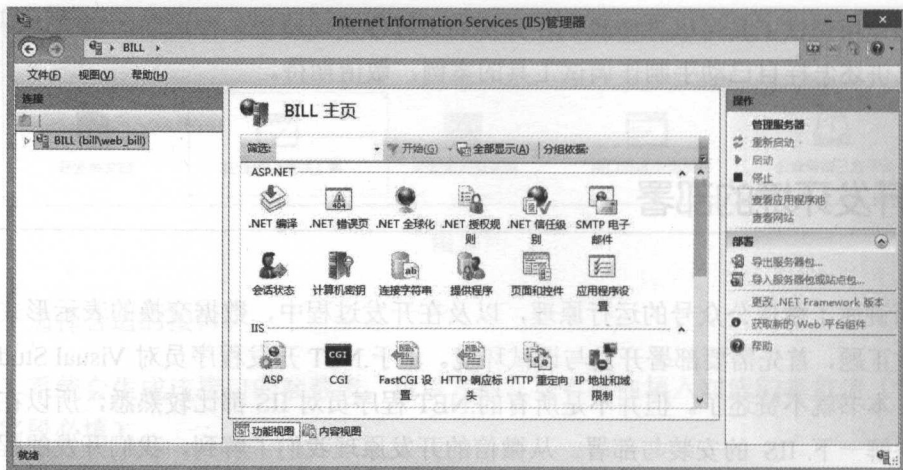


图 2-10

下面演示在 IIS 中部署 Web 应用程序。

首先，右击“网站”，在弹出的快捷菜单中单击“添加网站”，进入“添加网站”对话框，如图 2-11 所示。

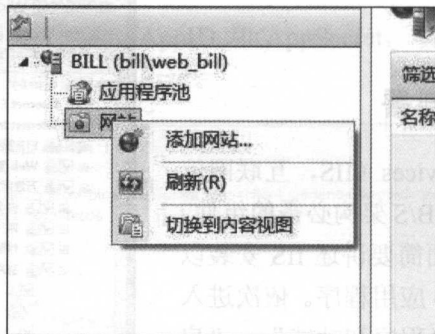


图 2-11

在如图 2-12 所示的对话框中，依次输入网站名称、应用程序池、物理路径、IP 地址、端口、主机名等信息。其中，“应用程序池”可以选择已经存在的，也可以创建一个新的，默认是创建一个与网站名称一致的应用程序池。“物理路径”可以是发布后的代码文件目录，也可以是开发过程中项目的文件目录，正式上线的时候建议选择发布后的目录。“IP 地址”可不选。“端口”默认是 80 端口，微信的项目必须为 80 端口，其他项目无限制。假如你的项目需要绑定域名的话，可将域名填写在“主机名”中，保存即可。

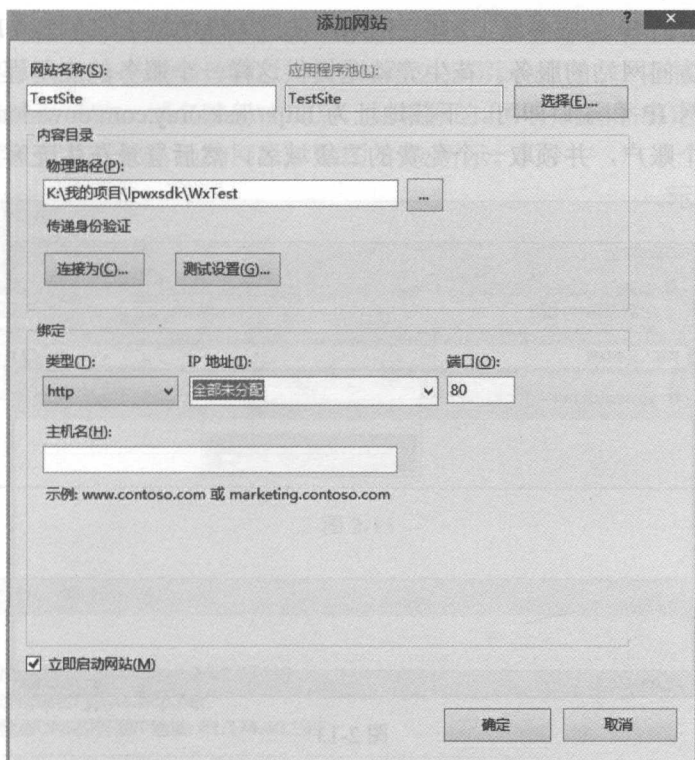


图 2-12

如果在新建网站时，没有设置 IP 地址，则本地可用“192.168.x.xxx”的内网 IP 访问，也可以用“127.0.0.1”或者“localhost”访问。如果绑定了 IP，则只能使用绑定的域名访问。然而，不管设置不设置 IP，这些访问方式都只能在内网访问。但如果没有公网 IP 的话，我们又如何让外网访问我们的 Web 应用程序呢？

2.2.2 花生壳域名映射与 ngrok

前面讲述了 IIS 的安装与网站的部署，但我们在开发微信的过程中需要将自己的本地计算机配置成外网可以访问的 Web 服务器，而花生壳与 ngrok 完美解决了该问题。

动态域名解析，简称 DDNS，可将用户的动态 IP 地址映射到一个固定的域名解析服务上。用户每次连接网络的时候，客户端程序就会通过信息传递把该主机的动态 IP 地址发送给位于服务商主机上的服务器程序，服务器程序负责提供 DNS 服务并实现动态域名解析。也就是说，DDNS 捕获用户每次变化的 IP 地址，然后将其与域名相对应，这样域名就可以



始终解析到非固定的 IP 的服务器上,互联网用户通过本地的域名服务器获取网站域名的 IP 地址,从而可以访问网站的服务。花生壳就是提供这样一个服务的应用程序。新版花生壳只需设置一下内网 IP 和端口即可。下载地址为 <http://hsk.oray.com/download/>。下载并安装后,需要注册一个账户,并领取一个免费的二级域名,然后登录花生壳客户端。登录后的界面如图 2-13 所示。

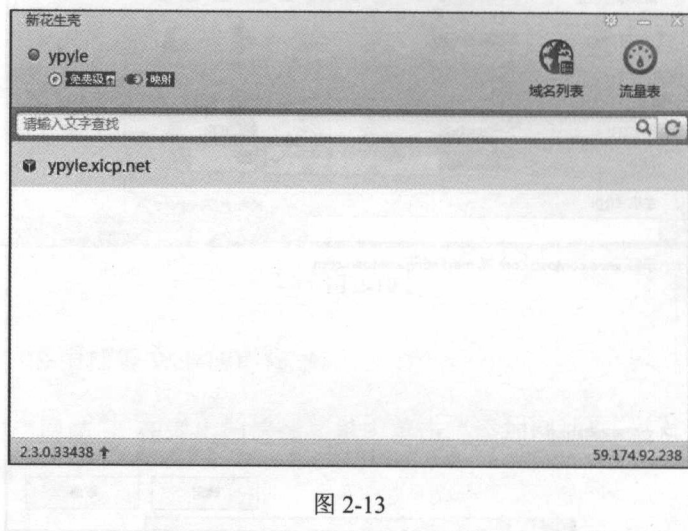


图 2-13

双击你的二级域名,进入“新花生壳管理”对话框,在这里可以编辑或添加映射。单击左上角的“添加映射”按钮,进入“添加映射”对话框,选择当前主机,或手动输入 IP 地址和端口号,勾选“使用外网 HTTP80 端口”,如图 2-14 所示。

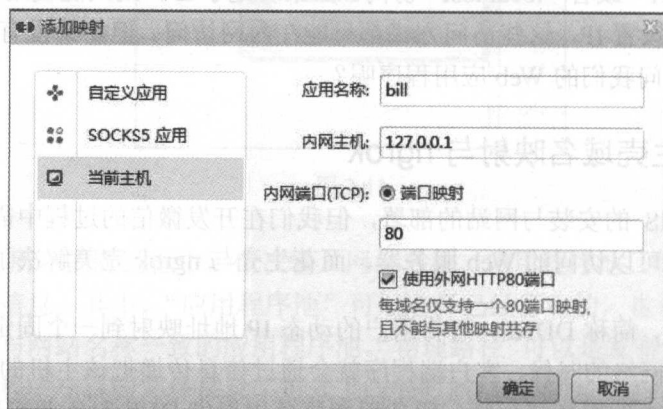


图 2-14



单击“确定”按钮，保存成功后，返回花生壳主窗口，右击你的域名，如图 2-15 所示，在弹出的快捷菜单中选择“域名诊断”，弹出如图 2-16 所示的窗口。在图 2-16 中，我们可以看到测试域名映射成功。域名映射成功后，如果你的 IIS 中有网站绑定了 80 端口，此时访问你的域名就可以打开你本地的网站，如图 2-17 所示。

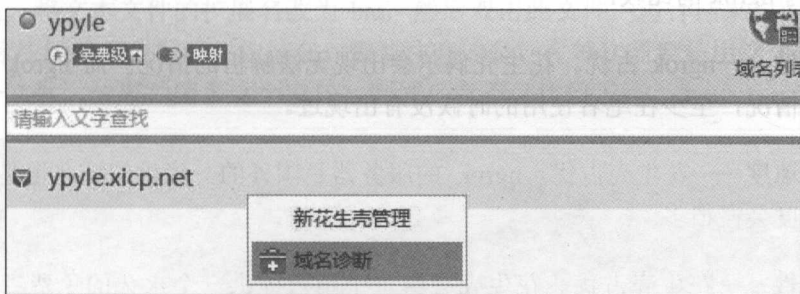


图 2-15

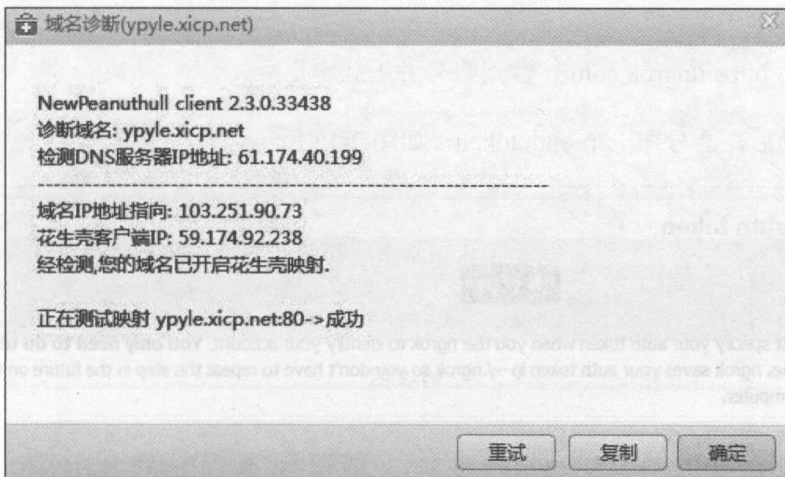


图 2-16

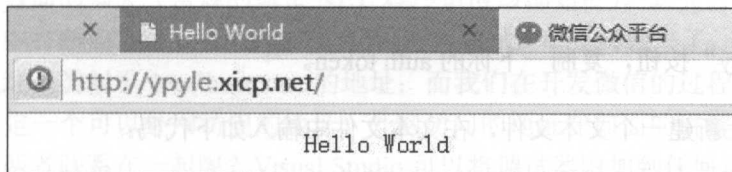


图 2-17



同花生壳一样，ngrok 也能实现花生壳的功能。花生壳是收费的；而免费的版本是需要抢使用资格的，并且免费的版本有时也会出现无法解析的情况。ngrok 则是免费使用的，我们只需注册一个账号，然后进行简单的设置即可正常使用。

花生壳与 ngrok 的比较：

- **稳定性**——ngrok 占优。花生壳偶尔会出现无法解析的情况；而 ngrok 没有出现过这种情况，至少在笔者使用的时候没有出现过。
- **访问速度**——花生壳占优。ngrok 的服务器是国外的，提供的域名也是国外的，访问速度会慢很多。
- **易用性**——花生壳占优。花生壳会给每个用户提供一个永久的免费二级域名；而 ngrok 提供的域名会改变，有的时候两三天就变一次。

使用方法

进入网站 <https://ngrok.com/>，注册账号并下载客户端。

注册成功后，会分配一个 auth token，如图 2-18 所示。

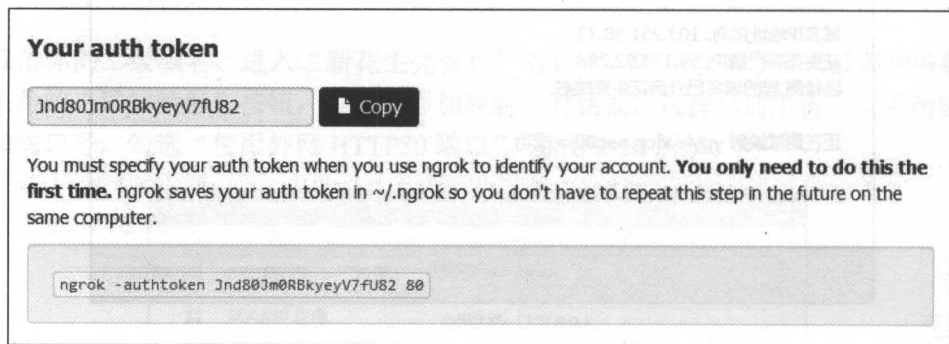


图 2-18

单击“Copy”按钮，复制一下你的 auth token。

在计算机上新建一个文本文件，在文本文件中输入如下代码：

```
cd 【ngrok 所在目录】
ngrok -authtoken 【你的 auth token】 【端口】
```




示例代码如下：

```
cd C:\ngrok
ngrok -authtoken i1Et1LlBr8QgcMaTe8hO 80
```

保存后，将文本文件的扩展名改为 bat，然后双击此文件，运行结果如图 2-19 所示。图 2-19 中标注的就是分配给你的域名。需要注意的是，在每次需要使用这个域名时，需要运行着这个文件。如果关闭了这个窗口，则域名就不可访问了。

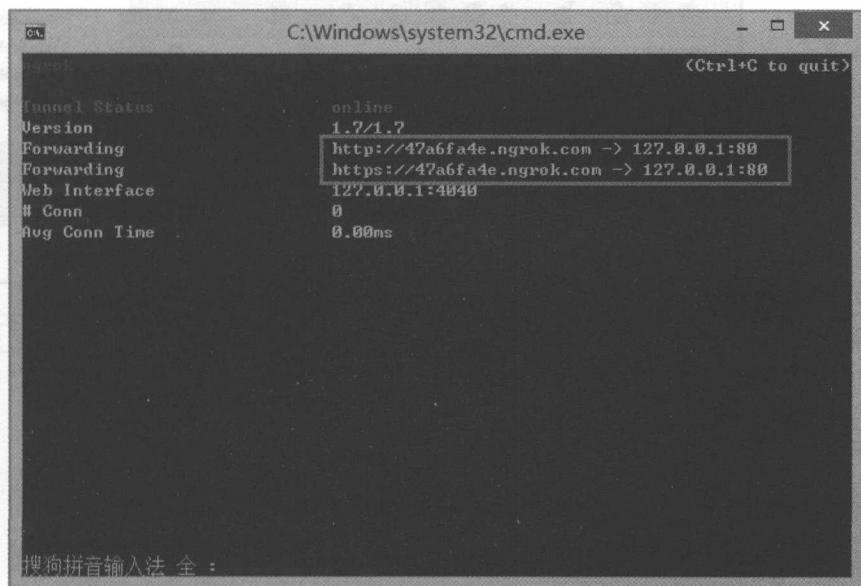


图 2-19

2.2.3 Visual Studio 本地调试

动态域名解析让我们的本地计算机成为可以外网访问的服务器，而大家都知道 Visual Studio 在调试方面的强大（常规的调试.NET 程序员应该都知道）。首先打好断点，也可以在调试的过程中打断点，然后按 F5 键，之后则进入调试。可是问题来了，这样调试的时候浏览器中的地址是类似于 localhost:xxx 的地址；而我们在开发微信的过程中，需要在微信公众平台绑定一个可以外网访问的地址。外网访问的地址在前面的讲解中我们已经解决了，但怎么让两者联系在一起呢？Visual Studio 可以将调试器附加到任何正在运行的进程，当我们的网站运行时，与之相关联的应用程序池会创建一个 w3wp.exe 的进程实例。所以，我们只需将 w3wp.exe 附加到调试器即可调试与我们本地计算机 80 端口绑定的请求。下面



是详细的操作过程。

首先，在 IIS 中新建一个网站，“物理路径”选择项目中 Web 应用程序的目录，并将端口设置为 80，如图 2-20 所示。“WxTest”是 Web 应用程序项目，从图 2-21 中可以看出，若“WxTest”的目录为“K:\我的项目\lpwxsdk\WxTest”，则新建网站时，物理路径为“K:\我的项目\lpwxsdk\WxTest”。

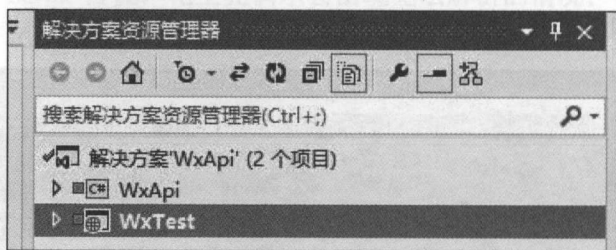


图 2-20

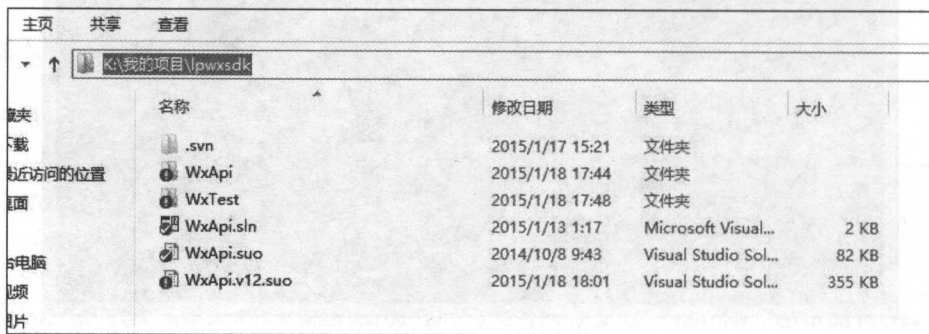


图 2-21

网站创建好后，启动动态域名解析（花生壳或 ngrok），访问绑定在 80 端口的域名。此时就会创建一个和刚刚新建网站相关联的 w3wp 进程。

然后以管理员身份运行 Visual Studio 并打开要调试的项目。依次单击菜单栏中的“调试”→“附加到进程”，或按快捷键“Alt+D+P”，打开“附加到进程”窗口，勾选窗口左下角的“显示所有用户进程”选项，并单击“刷新”按钮，在可用进程列表中找到和你的网站相关联的 w3wp.exe 进程，如图 2-22 所示。在“用户名”一列显示的是要调试网站相关联的应用程序池名称。如果同时启动了多个网站，而每个网站使用的都是独立的应用程序池，就会有多个 w3wp.exe 进程。此时，我们就只能根据用户名来区分不同的网站了。选



择好进程，单击“附加”按钮即可进入调试。

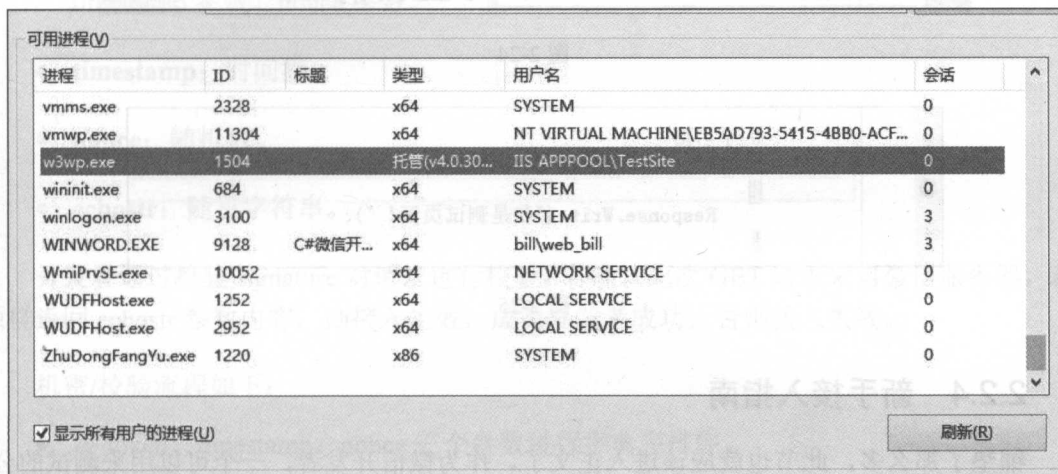


图 2-22 可用进程

如果 Visual Studio 启动时不是以管理员身份运行的，“用户名”一栏则是空的，并且单击“附加”按钮时，会弹出如图 2-23 所示的对话框。

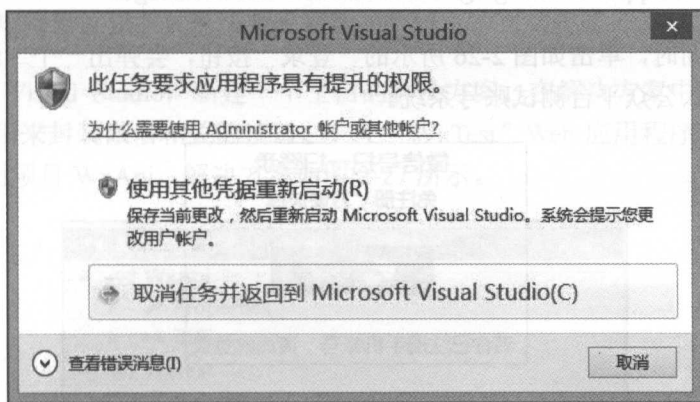


图 2-23 提升权限提示

单击“使用其他凭据重新启动”项后，Visual Studio 将以管理员身份运行。

进入调试后，我们就可以打个断点看看效果了。如图 2-24 和图 2-25 所示，当我们访问网址 <http://ypyle.xicp.net/index.aspx> 时，即会进入 index.aspx 页面的断点。

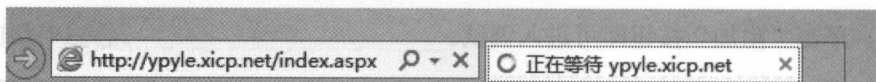


图 2-24

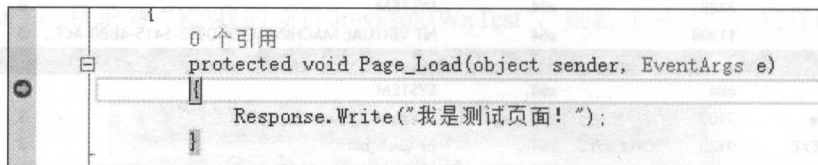


图 2-25

2.2.4 新手接入指南

铺垫了那么多，此节也就应该进入正文了。作为微信开发者，一个可以用来测试的公众号是必不可少的。为了方便开发者测试，微信提供了测试号。测试号拥有大部分的高级功能权限，在前期学习的过程中，可以使用测试账号进行开发。如果需要开发微信支付、微信小店之类的，则必须使用认证了的服务号。申请测试号的地址如下：

<http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=sandbox/login>

进入申请页面时，单击如图 2-26 所示的“登录”按钮，会弹出一个二维码的页面，扫描二维码即可进入公众平台测试账号系统。

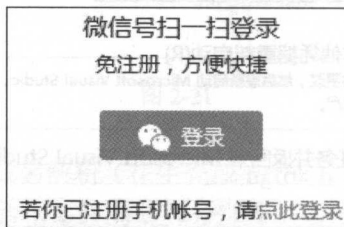


图 2-26

在 2.1 节中，讲述了微信公众号的原理，我们知道接入微信的时候，需要一个用来处理微信端请求的服务器端程序。本节就来讲解接入的步骤与接入程序的规则。

首先，开发者在接入时，微信服务器将发送 GET 请求到我们的 Web 程序中，我们需要提供一个 URL 来处理微信的请求。微信请求 URL 时，将携带四个参数。



- **signature**: 微信加密签名, signature 结合了开发者填写的 token 参数和请求中的 timestamp 参数、nonce 参数。
- **timestamp**: 时间戳。
- **nonce**: 随机数。
- **echostr**: 随机字符串。

开发者通过检验 signature 对请求进行校验。若确认此次 GET 请求来自微信服务器, 请原样返回 echostr 参数内容, 则接入生效, 成为开发者成功; 否则接入失败。

机密/校验流程如下:

- 将 token、timestamp、nonce 三个参数进行字典序排序。
- 将三个参数字符串拼接成一个字符串进行 SHA1 加密。
- 开发者获得加密后的字符串, 转换成小写与 signature 对比, 相等则标识该请求来源于微信。

知道了校验规则后, 我们只需按照流程编写代码, 即可实现接入。

首先, 打开 Visual Studio, 新建一个空白的解决方案。在解决方案中, 添加一个类库项目 “WxApi” 用来封装微信相关的接口、一个 “WxTest” Web 应用程序用来测试接口, 以及 WxTest 引用项目 WxApi。解决方案如图 2-27 所示。

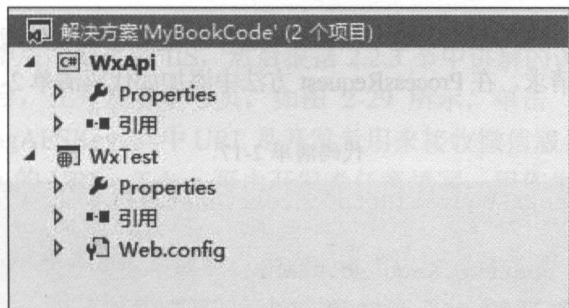


图 2-27

将 2.1.1 节中讲到的 Utils 类复制到 WxApi 项目中, 将此类作为通用的工具类使用。然



后在 WxApi 项目中添加一个封装基础服务的类 BaseServices。在 BaseServices 类中添加一个方法，如代码清单 2-16 所示。

代码清单 2-16

```
public static bool ValidUrl (string token)
{
    //获取参与校验的参数
    var signature = HttpContext.Current.Request.QueryString["signature"];
    var timestamp = HttpContext.Current.Request.QueryString["timestamp"];
    var nonce = HttpContext.Current.Request.QueryString["nonce"];
    string[] temp = { token, timestamp, nonce };
    Array.Sort(temp); //字典序排序
    var tempstr = string.Join("", temp); //拼接字符串
    // SHA1 加密
    var tempsign = FormsAuthentication.
        HashPasswordForStoringInConfigFile(tempstr, "SHA1").ToLower();
    if (tempsign==signature)
    {
        var echostr = HttpContext.Current.Request.QueryString["echostr"];
        HttpContext.Current.Response.Write(echostr);
        return true;
    }
    return false;
}
```

注：上述代码中的 HttpContext 类需要添加 System.Web 引用。

封装好用户验证接入的方法后，在 WxTest 项目中新建一个一般处理程序 wx.ashx，此页面用于处理微信服务器的请求。在 ProcessRequest 方法中添加如代码清单 2-17 所示的代码。

代码清单 2-17

```
public void ProcessRequest(HttpContext context)
{
    var url = context.Request.RawUrl;
    if (context.Request.HttpMethod=="GET")
    {
        BaseServices.ValidUrl("qqq");
    }
}
```



其中，方法体的第一行代码是调试使用，获取当前请求的原始 URL，在调试的过程中，可以一目了然地看到微信服务器请求的参数，如图 2-28 所示。if 语句中判断当前的请求是否是 GET 请求。如果是，则可以判定当前请求是接入请求，即可执行 `BaseServices.ValidUrl` 方法，验证请求是否合法，方法中的参数是自定义的，但要求在申请接入的时候，微信端填写的要和这里传的一样。如果是 POST，则说明是接入成功后，微信在发消息给我们的服务器。

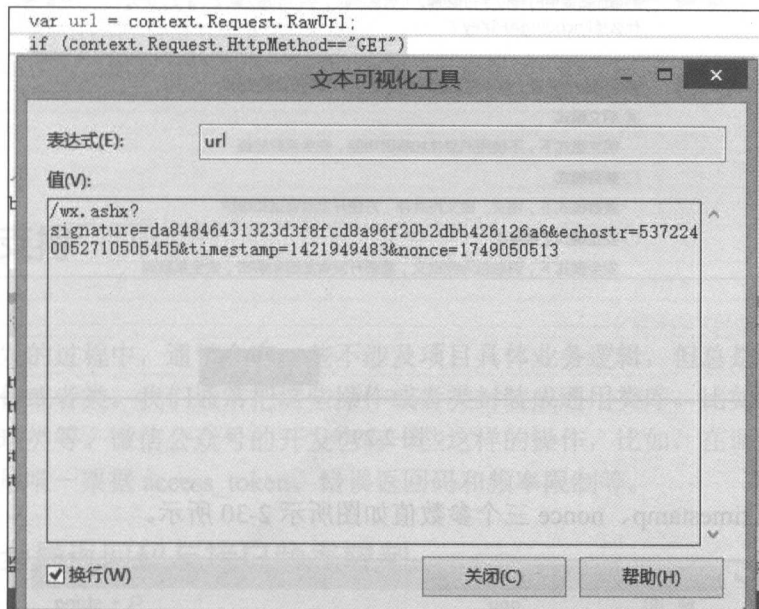


图 2-28

处理程序编写完毕后，部署到 IIS，然后根据 2.2.3 节中讲解的调试方法，进入调试模式。登录微信管理后台，在开发者中心页，如图 2-29 所示，单击“修改配置”按钮填写 URL、Token 和 EncodingAESKey。其中 URL 是开发者用来接收微信服务器数据的接口 URL，也就是上文中 wx.ashx 的 URL。Token 可由开发者任意填写，用作生成签名（该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性，需和代码中的 Token 保持一致）。EncodingAESKey 由开发者手动填写或随机生成，将用作消息体加解密密钥。同时，开发者可选择消息加解密方式：明文模式、兼容模式和安全模式。模式的选择与服务器配置在提交后都会立即生效。加解密方式的默认状态为明文模式。关于加解密的方式将在后面章节讲述。此处我们先选择明文模式，单击“提交”按钮后，微信发送 GET 请求到 URL 中，此时将进入我们的断点。



URL 必须以http://开头，目前支持80端口。

Token 必须为英文或数字，长度为3-32字符。
什么是Token？

EncodingAESKey 消息加密密钥由43位字符组成，可随机修改，字符范围为A-Z，a-z，0-9。
什么是EncodingAESKey？

消息加解密方式 请根据业务需要，选择消息加解密类型，启用后将立即生效

☒ 明文模式
明文模式下，不使用消息体加解密功能，安全系数较低

☐ 兼容模式
兼容模式下，明文、密文将共存，方便开发者调试和维护

☐ 安全模式（推荐）
安全模式下，消息包为纯密文，需要开发者加密和解密，安全系数高

图 2-29

signature、timestamp、nonce 三个参数值如图所示 2-30 所示。

temp	{string[3]}	string[]
[0]	"qqq"	Q - string
[1]	"1421950872"	Q - string
[2]	"487306513"	Q - string

图 2-30

经过排序、加密、转换成小写后，最终得到的签名如图 2-31 所示。

temp	{string[3]}
tempsign	"8caeb6e5403b431a14f01b2eb857e57365d4af6a"
signature	"8caeb6e5403b431a14f01b2eb857e57365d4af6a"

图 2-31

可以看出，根据签名规则生成的签名与微信发送来的签名一致，所以接入成功。

第3章 微信对话服务

3.1 基础支持

在软件开发的过程中，通常会有一些不涉及项目具体业务逻辑，但总是活跃在项目中各个角落的操作或者类。我们通常把这些操作或者类封装成通用类库。比如通用数据访问类、字符串处理类等。微信公众号的开发也有一些这样的操作，比如，在调用任何微信接口都使用的全局唯一票据 `access_token`、错误返回码和频率限制等。

3.1.1 全局返回码与接口频率限制

在正式讲解接口调用前，有必要先了解一下微信对于接口频率限制调用的一些限制，以及处理接口调用错误的机制。

为了防止公众号的程序错误而引发微信服务器负载异常，默认情况下，每个公众号调用接口都不能超过一定的限制。当超过一定限制时，调用对应接口会收到如下错误返回码（由于接口调用的限制并不是不变的，官方有的时候也会根据需要进行调整，因此关于接口频率限制说明请参阅官方文档，本书就不赘述了）：

```
{"errcode":45009,"errmsg":"api freq out of limit"}
```

公众号每次调用接口时，可能获得正确或错误的返回码，开发者可以根据返回码信息调试接口，排查错误。

全局返回码说明如表 3-1 所示。



表 3-1

返 回 码	说 明
-1	系统繁忙，此时请开发者稍候再试
0	请求成功
40001	获取 access_token 时 AppSecret 错误，或者 access_token 无效。请开发者认真比对 AppSecret 的正确性，或查看是否正在为恰当的公众号调用接口
40002	不合法的凭证类型
40003	不合法的 openid。请开发者确认 openid（该用户）是否已关注公众号，或是否是其他公众号的 openid
40004	不合法的媒体文件类型
40005	不合法的文件类型
40006	不合法的文件大小
40007	不合法的媒体文件 id
40008	不合法的消息类型
40009	不合法的图片文件大小
40010	不合法的语音文件大小
40011	不合法的视频文件大小
40012	不合法的缩略图文件大小
40013	不合法的 AppID。请开发者检查 AppID 的正确性，避免异常字符，注意大小写
40014	不合法的 access_token。请开发者认真比对 access_token 的有效性（如是否过期），或查看是否正在为恰当的公众号调用接口
40015	不合法的菜单类型
40016	不合法的按钮个数
40017	不合法的按钮个数
40018	不合法的按钮名字长度
40019	不合法的按钮 KEY 长度
40020	不合法的按钮 URL 长度
40021	不合法的菜单版本号
40022	不合法的子菜单级数
40023	不合法的子菜单按钮个数
40024	不合法的子菜单按钮类型
40025	不合法的子菜单按钮名字长度
40026	不合法的子菜单按钮 KEY 长度
40027	不合法的子菜单按钮 URL 长度
40028	不合法的自定义菜单使用用户
40029	不合法的 oauth_code
40030	不合法的 refresh_token
40031	不合法的 openid 列表



续表

返回码	说 明
40032	不合法的 openid 列表长度
40033	不合法的请求字符, 不能包含 \uxxxx 格式的字符
40035	不合法的参数
40038	不合法的请求格式
40039	不合法的 URL 长度
40050	不合法的分组 id
40051	分组名字不合法
41001	缺少 access_token 参数
41002	缺少 AppID 参数
41003	缺少 refresh_token 参数
41004	缺少 secret 参数
41005	缺少多媒体文件数据
41006	缺少 media_id 参数
41007	缺少子菜单数据
41008	缺少 oauth code
41009	缺少 openid
42001	access_token 超时, 请检查 access_token 的有效期
42002	refresh_token 超时
42003	oauth_code 超时
43001	需要 GET 请求
43002	需要 POST 请求
43003	需要 HTTPS 请求
43004	需要接收者关注
43005	需要好友关系
44001	多媒体文件为空
44002	POST 的数据包为空
44003	图文消息内容为空
44004	文本消息内容为空
45001	多媒体文件大小超过限制
45002	消息内容超过限制
45003	标题字段超过限制
45004	描述字段超过限制
45005	链接字段超过限制
45006	图片链接字段超过限制
45007	语音播放时间超过限制



续表

返回码	说 明
45008	图文消息超过限制
45009	接口调用超过限制
45010	创建菜单个数超过限制
45015	回复时间超过限制
45016	系统分组，不允许修改
45017	分组名字过长
45018	分组数量超过上限
46001	不存在媒体数据
46002	不存在的菜单版本
46003	不存在的菜单数据
46004	不存在的用户
47001	解析 JSON/XML 内容错误
48001	api 功能未授权。请确认公众号已获得该接口。可以在公众平台官网——开发者中心页中查看接口权限
50001	用户未授权该 api
61451	参数错误 (invalid parameter)
61452	无效客服账号 (invalid kf_account)
61453	客服账号已存在 (kf_account existed)
61454	客服账号名长度超过限制(仅允许 10 个英文字符, 不包括@及@后的公众号的微信号) (invalid kf_account length)
61455	客服账号名包含非法字符 (仅允许英文+数字) (illegal character in kf_account)
61456	客服账号个数超过限制 (10 个客服账号) (kf_account count exceeded)
61457	无效头像文件类型 (invalid file type)
61450	系统错误 (system error)
61500	日期格式错误
61501	日期范围错误

表 3-1 列出了所有的错误返回码及其对应的说明信息。但当调用接口发生错误时，返回的失败信息却是类似于图 3-1 的返回信息，直观上看不出太明确的错误，我们只能根据 errcode 到表 3-1 中找到对应的错误说明。

```
{"errcode":40013,"errmsg":"invalid appid"}
```

图 3-1

然而，根据返回的错误码人工去表中对比并不是程序员的思维与做法，程序员的思维就应该是使用程序的方式自动查找对应的错误详情。实现过程如下。



首先，从微信公众平台的开发文档中复制出表中的文本，在本地计算机上新建一个文本文件 CodeInfo.txt，将复制的文本粘贴到文本文件中，如图 3-2 所示。

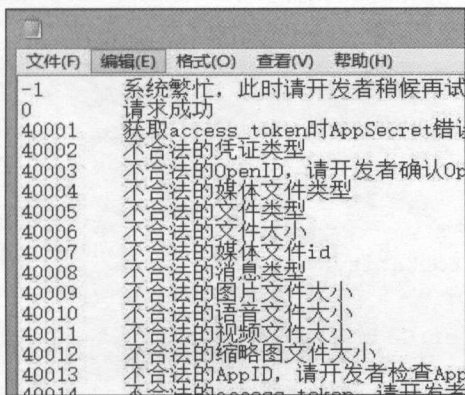


图 3-2

然后在 WxApi 项目中，新建一个资源文件 Code，如图 3-3 所示。

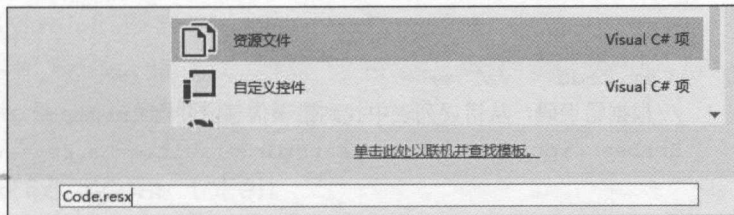


图 3-3

按照图 3-4 所示，先选择“文件”类型，再单击“添加资源”项，将 CodeInfo 添加到资源文件中。

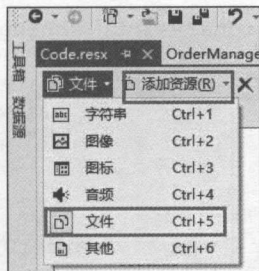


图 3-4



随后在 WxApi 项目中新建一个文件夹 ReceiveEntity, 用来存放请求接口时返回的实体。在 ReceiveEntity 文件夹中新建一个类 ErrorEntity, 如代码清单 3-1 所示。

代码清单 3-1

```
namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// 错误信息实体
    /// </summary>
    public class ErrorEntity
    {
        public int _errCode { get; set; }
        /// <summary>
        /// 错误编码
        /// </summary>
        public int ErrCode {
            get { return _errCode; }
            set
            {
                _errCode = value;
                //根据错误码, 从错误列表中找到错误信息, 并给 ErrDescription 赋值
                ErrDescription = ErrList.FirstOrDefault(e=>e.Key==value).Value;
            }
        }
        /// <summary>
        /// 错误描述
        /// </summary>
        public string ErrDescription { get; set; }

        private static Dictionary<int, string> _errorDic;

        public static Dictionary<int, string> ErrList
        {
            get
            {
                if (_errorDic != null && _errorDic.Count > 0)
                    return _errorDic;
                _errorDic = new Dictionary<int, string>();
            }
        }
    }
}
```




```
var temp = Code.CodeInfo.Split(new char[]{'\r', '\n'},
    StringSplitOptions.RemoveEmptyEntries);
foreach (var strArr in temp.Select(str => str.Split('\t')))
{
    _errorDic.Add(int.Parse(strArr[0]), strArr[1]);
}
return _errorDic;
}
}
}
```

ErrorEntity 类有一个静态的只读的 Dictionary<int, string>列表 ErrList, 用于保存所有错误信息实体。当获取 ErrList 的值时, 首先判断其值是否为空。如果不为空则直接返回; 如果为空, 则加载资源文件中的文本, 并转换成 ErrList 列表再返回。这里之所以使用静态变量来保存错误列表, 是因为错误列表并不会改变, 当列表初始化后, 就不需要再次初始化了, 提高了性能。还有两个实例属性 ErrCode 和 ErrDescription, 分别表示的是错误代码与错误描述。当给 ErrCode 赋值时, 会根据 ErrCode 的值, 从 ErrList 中找到与之对应的错误信息, 并赋值给 ErrDescription。所以在给 ErrorEntity 的实例赋值时, 只需给 ErrCode 赋值即可, ErrDescription 会自动找到关联项。

3.1.2 获取 access_token

在正式学习本节内容之前, 需要做一个准备工作。因为微信的接口大部分都是将指定格式的 JSON 消息 POST 给指定的接口地址或直接 GET 请求响应的接口地址, 所以为了使我们专注于接口的开发, 可以将 2.1.1 节 (HTTP 请求与响应) 中所述的 HttpGet 和 HttpPost 跟 JSON 的解析相关的操作进一步封装 (见代码清单 3-2)。

代码清单 3-2

```
/// <summary>
/// 发起 POST 请求, 并获取请求返回值
/// </summary>
/// <typeparam name="T">返回值类型</typeparam>
/// <param name="obj">数据实体</param>
/// <param name="url">接口地址</param>
public static T PostResult<T>(object obj, string url)
```




```
{
    //序列化设置
    var setting = new JsonSerializerSettings();
    //解决枚举类型序列化时,被转换成数字的问题
    setting.Converters.Add(new StringEnumConverter());
    var retdata = HttpPost(url, JsonConvert.SerializeObject(obj, setting));
    return JsonConvert.DeserializeObject<T>(retdata);
}
/// <summary>
/// 发起 postForm 请求, 并获取请求返回值
/// </summary>
/// <typeparam name="T">返回值类型</typeparam>
/// <param name="formEntities">数据实体</param>
/// <param name="url">接口地址</param>
public static T PostFormResult<T>(List<FormEntity> formEntities,
    string url)
{
    var retdata = HttpPostForm(url, formEntities);
    return JsonConvert.DeserializeObject<T>(retdata);
}
/// <summary>
/// 发起 GET 请求, 并获取请求返回值
/// </summary>
/// <typeparam name="T">返回值类型</typeparam>
/// <param name="url">接口地址</param>
public static T GetResult<T>(string url)
{
    var retdata = HttpGet(url);
    return JsonConvert.DeserializeObject<T>(retdata);
}
```

`access_token` 是公众号的全局唯一票据, 公众号调用各接口时都需要使用 `access_token`。由于 `access_token` 的有效期目前只有两个小时, 而获取 `access_token` 的次数每日是有限额的, 所以我们不能每次需要时都调用一次。为了保证 `access_token` 一直可用, 在获取到 `access_token` 之后, 需要将其保存起来。如果在 `access_token` 过期前没有刷新, 则接口调用就会失败, 重复获取 `access_token` 将导致上次获取的 `access_token` 失效。获取 `access_token` 的接口地址如下: https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&AppID=AppID&secret=AppSecret。HTTP 请求方式是 GET。



接口中的 AppID 是第三方用户唯一凭证，每一个公众号都有一个唯一的 AppID。secret 是配合 AppID 使用的，是唯一凭证密钥。在调用此接口时，只需将接口中的 AppID 和 AppSecret 替换成我们自己的，然后发送 GET 请求。正常情况下，微信会返回下述 JSON 数据包：

```
{"access_token":"ACCESS_TOKEN","expires_in":7200}
```

其中，access_token 就是获取到的凭证；expires_in 则是凭证的有效时间，单位为秒。所以在编写代码时，我们需要一个对象来存储 access_token。这个对象包含两个属性：一个表示 access_token 字符串，一个表示过期时间。

当调用错误时，微信服务器会返回错误码等信息，JSON 数据包示例如下（该示例表示 AppID 无效）：

```
{"errcode":40013,"errmsg":"invalid appid"}
```

下面是代码实现的步骤。

在 ReceiveEntity 文件夹中新建一个类 AccessToken，并继承 ErrorEntity 类，方便在调用失败时，返回错误信息给开发者，如代码清单 3-3 所示。

代码清单 3-3

```
/// <summary>
/// 全局票据实体
/// </summary>
public class AccessToken : ErrorEntity
{
    /// <summary>
    /// AccessToken 的值
    /// </summary>
    public string access_token { get; set; }
    /// <summary>
    /// 过期时间
    /// </summary>
    public DateTime ExpirationTime { get; set; }
    private int _expires_in;
    /// <summary>
    /// 凭证有效时间，单位：秒
    /// </summary>
    public int expires_in
```



```
{
    set
    {
        ExpirationTime = DateTime.Now.AddSeconds(value / 2);
        _expires_in = value;
    }
}
```

在类 `AccessToken` 中, `expires_in` 属性设置为只写。在给 `expires_in` 属性赋值时, 将当前时间加上 `expires_in/2` 秒, 并给属性 `ExpirationTime` 赋值。这里之所以除以 2, 是因为 `expires_in` 的值默认是 7200 秒, 也就是两个小时, `access_token` 每天的调用次数为 2000 次 (订阅号为 2000 次, 服务号为 100 000 次), 哪怕一个小时调用一次, 一天也只有 24 次, 所以在这个时间的基础上, 我除了 2, 这样既保证了极端情况下 `access_token` 有效, 也不会因为超出调用限额, 导致调用失败。

实体创建完毕后, 在 `BaseServices` 类中, 添加一个方法 `GetAccessToken`, 如代码清单 3-4 所示。

代码清单 3-4

```
public static AccessToken GetAccessToken(string appid, string appSecret)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid={0}&secret={1}", appid, appSecret);
    return Utils.GetResult<AccessToken>(url);
}
```

`BaseServices` 类中的 `GetAccessToken` 方法只是封装了获取 `access_token` 的方法, 但并没有将 `access_token` 进行缓存, 为了不限制 `GetAccessToken` 方法的扩展性, 缓存 `access_token` 的操作将根据具体的业务逻辑进行缓存。下面的实例讲解的是在支持多公众号的项目中缓存 `access_token`。

首先, 在 `WxTest` 项目中添加一个类 `AccessTokenBox`, 在类中添加一个 `string` 类型的属性 `AppId`, 用来和获取到的 `access_token` 对应; 一个 `AccessToken` (要导入 `WxApi.ReceiveEntity` 命名空间) 类型的属性 `Token`, 用来保存获取到的 `AccessToken` 对象。之后, 添加一个私有的、静态的 `AccessTokenBox` 列表字段 `_boxs`, 用来缓存项目中所有公众号的 `access_token`。



然后, 在 `AccessTokenBox` 中添加一个方法 `GetTokenValue`, 方法有两个参数, 分别是 `appid` 和 `appSecret`. `GetTokenValue` 业务逻辑如下。

- 判断 `boxes` 是否为空。若为空, 则进行实例化; 若不为空, 则筛选出 `access_token` 未过期的 `AccessTokenBox` 列表。
- 根据 `appid` 从筛选出来的列表中查找对应的 `AccessTokenBox`, 并返回 `access_token` 字符串。如果未找到, 就说明此公众号的 `access_token` 已过期或尚未获取过, 则调用类 `BaseServices` 的 `GetAccessToken` 进行获取。
- 如果获取成功, 则将实例化 `AccessTokenBox` 对象, 将获取到的 `AccessToken` 对象赋值给 `Token` 属性, 并返回获取到的 `access_token` 字符串。如果获取失败, 则可根据具体的业务逻辑进行操作。

具体实现如代码清单 3-5 所示。

代码清单 3-5

```
using System;
using System.Collections.Generic;
using System.Linq;
using WxApi;
using WxApi.ReceiveEntity;
namespace WxTest
{
    public class AccessTokenBox
    {
        public string AppId { get; set; }
        public AccessToken Token { get; set; }
        private static List<AccessTokenBox> _boxes;
        public static string GetTokenValue(string appid, string appSecret)
        {
            _boxes = (_boxes == null ? new List<AccessTokenBox>() :
            _boxes.Where(b=>b.Token.ExpirationTime>DateTime.Now).ToList());
            var tempat = _boxes.FirstOrDefault(b => b.AppId == appid);
            if (tempat != null)
            {
                return tempat.Token.access_token;
            }
            var newAT = BaseServices.GetAccessToken(appid, appSecret);
```




```
if (!string.IsNullOrEmpty(newAT.access_token))
{
    _boxs.Add(new AccessTokenBox
    {
        AppId = appid,
        Token = newAT
    });
    return newAT.access_token;
}
else
{
    //此处可以写日志，保存错误信息
    return "";
}
}
```

在项目中，按代码清单 3-6 调用。

代码清单 3-6

```
var AccessToken = AccessTokenBox.GetTokenValue("你的appid", "你的secret");
```

3.1.3 获取微信服务器 IP 地址

在 2.2.4 节中，讲解了公众号与服务器接入相关的配置。在接入的过程中，服务器根据校验规则和自定义的 Token 以及接收到的参数来校验签名是否匹配。校验规则是微信开放的接口，任何人都可以获取。也可以根据校验规则生成请求的参数，但 Token 是用户自定义的，如果不知道自定义的 Token 的值，就无法通过校验，相对来说是比较安全的。但如果我们的接入地址与 Token 被黑客截取，黑客就可以假冒微信服务器来请求我们的服务器。基于安全因素考虑，腾讯开放了获取微信服务器的 IP 地址列表接口，开发者只需在服务器被请求时，获取当前请求的 IP 地址，然后在微信服务器的 IP 列表中查找是否有匹配项，以此来校验请求是否合法。获取微信服务器 IP 地址的接口如下：

https://api.weixin.qq.com/cgi-bin/getcallbackip?access_token=ACCESS_TOKEN

HTTP 请求方式：GET。



代码实现如下。

首先，在 `ReceiveEntity` 文件夹中新建类 `IpEntity`，此类继承 `ErrorEntity`，在类中添加一个字符串数组的属性 `ip_list`，如代码清单 3-7 所示。

代码清单 3-7

```
namespace WxApi.ReceiveEntity
{
    public class IpEntity : ErrorEntity
    {
        /// <summary>
        /// IP 列表
        /// </summary>
        public string[] ip_list { get; set; }
    }
}
```

然后在 `BaseServices` 类中新建一个方法 `GetIpArray`，如代码清单 3-8 所示。

代码清单 3-8

```
public static IpEntity GetIpArray(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/getcallbackip?access_token={0}", accessToken);
    return Utils.GetResult<IpEntity>(url);
}
```

最后，就可以校验请求的 IP 是否合法了。在 `WxTest` 项目的一般处理程序 `wx.ashx` 中，增加代码清单 3-9 所示的代码。

代码清单 3-9

```
public void ProcessRequest(HttpContext context)
{
    var ip = context.Request.UserHostAddress;
    var ipEntity = BaseServices.GetIpArray("你的 access_token");
    if (ipEntity != null && !ipEntity.ip_list.Contains(ip))
```




```
{
    context.Response.Write("非法请求");
    return;
}
if (context.Request.HttpMethod == "GET")
{
    BaseServices.ValidUrl("qqq");
}
else
{
    //微信服务器消息处理业务逻辑
}
```

其中，GetIpArray 方法中的参数需要调用 3.1.2 节中获取 access_token 的方法。另外需要注意的是，如果你是使用 2.2.2 节中讲的花生壳或者 ngrok 的方式调试的，则获取到的 IP 是 127.0.0.1，不是真正的微信服务器的 IP。在真实的服务器环境中获取到的是真正微信服务器的 IP，所以**建议在项目要上线的时候再加上上面的校验代码，调试的时候就不要加了**。在获取微信服务器 IP 列表后，建议参照获取 access_token 的方法将获取的 IP 缓存起来。虽然官方文档并没有说明此接口的调用限制，但也没有说明获取的 IP 列表会不会变化。所以为了项目的稳定与性能考虑，需要将 IP 列表缓存。限于篇幅，在此就不赘述了。

3.2 素材管理接口

微信公众平台中的素材指的是公众号给订阅用户发送消息的内容。公众号在给订阅用户发送消息之前需要先将素材上传。素材类型包括图片(image)、语音(voice)、视频(video)、缩略图(thumb)和图文(news)，其中图片和缩略图由于展示方式是一样的，所以可以理解为同一种类型的素材。

3.2.1 新增素材

为了更好地利用资源，根据素材的使用场景，微信又将素材分为临时素材和永久素材。对于临时素材，每个素材(media_id)会在开发者上传或粉丝发送到微信服务器 3 天后自动删除，以节省服务器资源。所以用户发送给开发者的素材，若开发者需要，应尽快下载到本地。素材在有效期内，每个 media_id 是可复用的。素材的格式大小等要求与公众平台



官网一致。具体是，图片大小不超过 2MB，支持 bmp、png、jpeg、jpg、gif 格式；语音大小不超过 5MB，长度不超过 60s，支持 mp3、wma、wav、amr 格式；视频大小不超过 20MB，支持 rm、rmvb、wmv、avi、mpg、mpeg、mp4 格式。新增临时素材的接口地址如下：

`https://api.weixin.qq.com/cgi-bin/media/upload?access_token=ACCESS_TOKEN&type=TYPE`

HTTP 请求方式：**POST/FORM**。

在接口地址中，type 表示的是媒体文件类型，包括图片、语音、视频和缩略图。FORM 表单的 ID 为 media。

除了 3 天就会失效的临时素材外，开发者有时还需要永久保存一些素材。新增的永久素材也可以在公众平台官网素材管理模块中看到。永久素材的数量是有上限的，请谨慎新增。图文消息素材和图片消息素材的上限为 5000，其他类型为 1000。另外，新增永久素材又被分为图文素材和其他素材。其他素材和临时素材支持的文件类型是一致的，也就是说临时素材不支持图文素材，新增其他类型永久素材的接口地址如下：

`http://api.weixin.qq.com/cgi-bin/material/add_material?access_token=ACCESS_TOKEN`

HTTP 请求方式：**POST/FORM**。

语音、图片、缩略图的调用方式，临时素材和永久素材是一致的。但新增永久视频素材需特别注意，在上传视频素材时，需要 POST 另一个表单，id 为 description，包含素材的描述新消息，内容格式为 JSON，格式如下：

```
{"title":VIDEO_TITLE, "introduction":INTRODUCTION}
```

为了方便管理和代码编写，我们可以把素材类型定义为枚举类型，如代码清单 3-10 所示。

代码清单 3-10

```
/// <summary>
/// 素材类型枚举
/// </summary>
public enum MaterialType
{
    /// <summary>
```



```
/// 图片 (image) : 2MB, 支持 bmp/png/jpeg/jpg/gif 格式
/// </summary>
image,
/// <summary>
/// 语音 (voice) : 5MB, 播放长度不超过 60s, 支持 mp3/wma/wav/amr 格式
/// </summary>
voice,
/// <summary>
/// 视频 (video) : 20MB, 支持 rm/rmvb/wmv/avi/mpg/mpeg/mp4 格式
/// </summary>
video,
/// <summary>
/// 缩略图 (thumb) : 64KB, 支持 jpg 格式
/// </summary>
thumb,
/// <summary>
/// 图文
/// </summary>
news
}
```

正常情况下, 新增临时素材成功后, 返回的 JSON 数据包如下:

```
{"type": "TYPE", "media_id": "MEDIA_ID", "created_at": 123456789}
```

新增永久素材成功后, 返回的 JSON 数据包如下:

```
{ "media_id": MEDIA_ID }
```

根据返回的 JSON 数据包创建实体类 UploadInfo, 如代码清单 3-11 所示。

代码清单 3-11

```
/// <summary>
/// 新增素材时, 返回的实体
/// </summary>
public class UploadInfo: ErrorEntity
{
    /// <summary>
    /// 媒体文件类型, 分别有图片 (image)、语音 (voice)、视频 (video) 和缩略图 (thumb,
```



```
    ///主要用于视频与音乐格式的缩略图)
    /// </summary>
    public string type { get; set; }
    /// <summary>
    /// 媒体文件上传后, 获取时的唯一标识
    /// </summary>
    public string media_id { get; set; }
    /// <summary>
    /// 媒体文件上传时间戳
    /// </summary>
    public string created_at { get; set; }
}
```

准备工作完成后, 就可以正式进入正题了。新建类 `MaterialLib`, 用于封装素材管理相关的接口。在类中新建方法 `Add`, 功能是新增素材, 并返回新增结果。由于上传临时素材和永久素材的处理方式基本相同, 唯一不同的是接口不同, 因此在 `Add` 方法中, 根据参数 `IsTemp` 来判断此次调用是新增临时素材还是永久素材, 默认为临时 (如代码清单 3-12 所示)。

代码清单 3-12

```
/// <summary>
/// 添加素材。临时素材的有效时间为 3 天
/// </summary>
/// <param name="filePath">服务器文件的物理路径, 可用 Request.MapPath 将虚拟路径
/// 转换为物理路径。也可为网络路径, 如: http://XXXX</param>
/// <param name="accessToken">调用凭据</param>
/// <param name="mediaType">媒体类型枚举</param>
/// <param name="IsTemp">是否是临时素材</param>
/// <param name="videotitle">永久视频素材标题</param>
/// <param name="videointroduction">永久视频素材描述</param>
public static UploadInfo Add(string filePath, string accessToken,
    MaterialType mediaType, bool IsTemp = true, string videotitle = "",
    String videointroduction = "")
{
    try
    {
        //临时素材接口
        var url =
```




```
"https://api.weixin.qq.com/cgi-bin/media/upload?access_token={0}&type={1}";
    if (!IsTemp)
    {
        //永久素材接口
        url =
        "http://api.weixin.qq.com/cgi-bin/material/add_material?access_token={0}&type={1}";
    }
    var formlist = new List<FormEntity>
    {
        new FormEntity { IsFile = true, Name = "media", Value = filePath }
    };
    if (mediaType == MaterialType.video && !IsTemp)
    {
        //新增视频素材的特殊处理
        var value = JsonConvert.SerializeObject(new { title = videotitle, introduction = videointroduction });
        formlist.Add(new FormEntity { IsFile = false, Name = "description", Value = value });
    }
    return Utils.PostFormResult<UpLoadInfo>(formlist, string.Format(url, accessToken, mediaType.ToString()));
}
catch (Exception e)
{
    return new UpLoadInfo
    {
        ErrCode = -2,
        ErrDescription = e.Message
    };
}
}
```

新增永久图文素材的接口地址如下：

https://api.weixin.qq.com/cgi-bin/material/add_news?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。



POST 数据示例如代码清单 3-13 所示。

代码清单 3-13

```
{
  "articles": [{
    "title": TITLE,
    "thumb_media_id": THUMB_MEDIA_ID,
    "author": AUTHOR,
    "digest": DIGEST,
    "show_cover_pic": SHOW_COVER_PIC(0 / 1),
    "content": CONTENT,
    "content_source_url": CONTENT_SOURCE_URL
  }],
  //若新增的是多图文素材，则此处应还有几段 articles 结构
}
```

从示例中可以看出，属性 `articles` 的类型是数组，数组的每一项由一个图文实体组成，包括标题 (`title`)、作者 (`author`)、内容 (`content`) 等，这些属性和公众平台后台添加图文消息是一一对应的。根据示例，在项目 `WxApi` 中添加文件夹 `SendEntity`，并创建图文项实体类 `Article`，如代码清单 3-14 所示。

代码清单 3-14

```
namespace WxApi.SendEntity
{
    /// <summary>
    /// 图文素材图文项实体
    /// </summary>
    public class Article
    {
        /// <summary>
        /// 图文消息的封面图片素材 id (必须是永久 mediaID)
        /// </summary>
        public string thumb_media_id { get; set; }
        /// <summary>
        /// 图文消息的作者
        /// </summary>
    }
```




```
public string author { get; set; }  
/// <summary>  
/// 图文消息的标题  
/// </summary>  
public string title { get; set; }  
/// <summary>  
/// 在图文消息页面单击“阅读原文”后的页面  
/// </summary>  
public string content_source_url { get; set; }  
/// <summary>  
/// 图文消息页面的内容，支持 HTML 标签，且此处会去除 JS  
/// </summary>  
public string content { get; set; }  
/// <summary>  
/// 图文消息的摘要，仅有单图文消息才有摘要，多图文此处为空  
/// </summary>  
public string digest { get; set; }  
/// <summary>  
/// 是否显示封面，1 为显示，0 为不显示  
/// </summary>  
public int show_cover_pic { get; set; }  
}  
}
```

代码清单 3-15 所示为新增永久图文消息的方法。

代码清单 3-15

```
public static UploadInfo AddArticle(List<Article> articles, string  
accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/cgi-bin/material/add_news?  
access_token={0}", accessToken);  
    return Utils.PostResult<UploadInfo>(new { articles = articles }, url);  
}
```

请注意，在图文消息的具体内容中，将过滤外部的图片链接。开发者可以通过调用上传图文消息内的图片接口上传图片，得到 URL，放到图文内容中使用。本接口所上传的图片不占用公众号的素材库中图片数量的 5000 个的限制，图片仅支持 jpg/png 格式，大小必



须在 1MB 以下。接口地址如下：

```
https://api.weixin.qq.com/cgi-bin/media/uploading?access_token=ACCESS_TOKEN
```

调用方式和 3.2.1 节中新增素材接口的方式是一致的；唯一不同的是，调用此接口后返回的是图片的 URL。

```
{
    "url":
"http://mmbiz.qpic.cn/mmbiz/gLOl7UPS6FS2xsypf378iaNhWacZlG1Up1ZYWEYfwvuU6Ont
96blroYs CNFwaRrSaKTPCUdBK9DgEHicsKwWCBRQ/0"
}
```

此接口返回的 URL 也可用于后续章节中的高级群发接口（见 3.5.1 节）。具体的代码实现就不在此赘述了，读者可参考 3.2.1 节中的新增素材接口或者参考本书随书源码。

3.2.2 根据 media_id 获取临时素材

获取临时素材的接口地址如下：

```
http://api.weixin.qq.com/cgi-bin/media/get?access_token=ACCESS_TOKEN&media_id=
MEDIA_ID
```

在调用的时候，只需传入合法的 access_token 和 media_id。可以使用代码实现 GET 请求，将素材下载到本地或服务器，也可以将拼接好的 URL 作为网络资源调用。代码清单 3-16 所示的功能是获取临时素材的 URL。

代码清单 3-16

```
public static string GetTempUrl(string mediaId, string accessToken)
{
    var url =
"http://api.weixin.qq.com/cgi-bin/media/get?access_token={0}&media_id={1}";
    return string.Format(url, accessToken, mediaId);
}
```

有的时候还需要将素材下载到本地或服务器。为了解决这个需求，可以将上述代码中得到的 URL 保存为流或者二进制，在需要下载或保存文件的地方调用会更方便。由于在下载文件时都需要指定文件名，但.NET 中提供的和流相关的类都没有文件名这一属性；因此，



为了更适用使用场景，新建类 `FileStreamInfo`，并继承 `MemoryStream`，如代码清单 3-17 所示。

代码清单 3-17

```
public class FileStreamInfo : MemoryStream
{
    /// <summary>
    /// 文件名
    /// </summary>
    public string FileName { get; set; }
}
```

在类 `Utils` 中添加方法 `DownloadStream`，用于下载 `FileStreamInfo` 类型的文件流，如代码清单 3-18 所示。

代码清单 3-18

```
public static void DownloadStream(FileStreamInfo stream)
{
    var bytes = stream.ToArray();
    HttpContext.Current.Response.ContentType = "application/octet-stream";
    HttpContext.Current.Response.AddHeader("Content-Disposition",
    "attachment;filename="+
    HttpContext.Current.Server.UrlDecode(stream.FileName));
    HttpContext.Current.Response.AddHeader("Content-Length",
    bytes.Length.ToString());
    HttpContext.Current.Response.BinaryWrite(bytes);
    HttpContext.Current.Response.Flush();
    HttpContext.Current.Response.End();
}
```

在类 `Utils` 中添加方法 `DownloadFile`，功能是根据文件的物理路径或网络路径下载文件，如代码清单 3-19 所示。

代码清单 3-19

```
/// <summary>
/// 下载文件
/// </summary>
```



```
/// <param name="fileUrl">文件路径, 可为网络资源, 也可以是文件物理路径</param>
/// <param name="fileName">文件名</param>
public static void DownLoadFile(string fileUrl, string fileName)
{
    using (var client = new WebClient())
    {
        //将网络资源下载为二进制数据
        var bytes = client.DownloadData(fileUrl);
        using (var fsi = new FileStreamInfo())
        {
            //将二进制数据写入文件流
            fsi.Write(bytes, 0, bytes.Length);
            fsi.FileName = fileName;
            DownLoadSteam(fsi);
        }
    }
}
```

3.2.3 根据 media_id 获取永久素材

在新增了永久素材后, 开发者可以根据 media_id 来获取永久素材, 需要时也可以保存到本地。获取永久素材也可以获取公众号在公众平台官网素材管理模块中新建的图文消息、语音、视频等素材 (但需要先通过获取素材列表来获取素材的 media_id)。接口地址如下:

https://api.weixin.qq.com/cgi-bin/material/get_material?access_token=ACCESS_TOKEN

和获取临时素材的区别是, 永久素材需要使用 POST 请求。调用示例如下:

```
{"media_id":MEDIA_ID}
```

如果请求的素材为图文消息, 则响应的数据包如代码清单 3-20 所示。

代码清单 3-20

```
{
  "news_item":
  [
    {
```




```
"title":TITLE,
"thumb_media_id":THUMB_MEDIA_ID,
"show_cover_pic":SHOW_COVER_PIC(0/1),
"author":AUTHOR,
"digest":DIGEST,
"content":CONTENT,
"content_source_url":CONTENT_SOURCE_URL
},
//多图文消息有多篇文章
]
```

如果返回的是视频消息素材，则响应的内容如下所示：

```
{
  "title":TITLE,
  "description":DESCRIPTION,
  "down_url":DOWN_URL,
}
```

若是其他类型的素材消息，则响应的直接为素材的内容，开发者可以自行保存为文件。因为除了图文素材和视频素材外，其他类型的素材都是媒体文件类型的，所以，这里可以理解为其为其他素材的响应应该为素材的文件流。

在代码的实现过程中，首先获取图文和视频素材是比较常规的，直接新建实体类后，调用 `Utils.PostResult` 方法即可。代码清单 3-21 和代码清单 3-22 分别是视频素材和图文素材的实体类。

代码清单 3-21

```
/// <summary>
/// 视频素材实体类
/// </summary>
public class MaterialVideo:ErrorEntity
{
    public string title { get; set; }
    public string description { get; set; }
    public string down_url { get; set; }
}
```




代码清单 3-22

```
/// <summary>
/// 获取永久素材时，当请求的素材为图文消息时，返回的实体
/// </summary>
public class MaterialNews:ErrorEntity
{
    /// <summary>
    /// 多图文列表
    /// </summary>
    public List<Article> news_item { get; set; }
}
```

实体创建完毕后，在 **MaterialLib** 类中分别创建获取图文素材和视频素材的方法，如代码清单 3-23 所示。

代码清单 3-23

```
/// <summary>
/// 获取永久图文素材
/// </summary>
public static MaterialNews GetNews(string mediaId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/material/
            get_material?access_token={0}", accessToken);
    return Utils.PostResult<MaterialNews>
        (url, JsonConvert.SerializeObject(new { media_id = mediaId }));
}
/// <summary>
/// 获取永久视频素材
/// </summary>
public static MaterialVideo GetVideo(string mediaId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/material/
            get_material?access_token={0}", accessToken);
    return Utils.PostResult<MaterialVideo>
        (url, JsonConvert.SerializeObject(new { media_id = mediaId }));
}
```



由于通过调用接口获取图文素材和视频素材返回的都是 JSON 字符串，因此处理起来比较简单。但由于通过调用接口获取其他素材（语音、图片等）返回的是文件的内容，处理起来相对比较麻烦。这里我们可以使用 WebClient 先获取文件的二进制，然后转换成文件流，最后再做响应的处理。具体实现如下。

首先，在 Utils 类中添加如下代码：

```
public static void DownloadByPost(string url, string data, Stream stream)
{
    using (var webclient = new WebClient())
    {
        var retdata = webclient.UploadData(url,
            "POST", Encoding.UTF8.GetBytes(data));
        stream.Write(retdata, 0, retdata.Length);
    }
}
```

然后，在 MaterialLib 类中添加调用上述方法的代码：

```
/// <summary>
/// 获取除了视频、图文素材之外的其他素材
/// </summary>
public static void GetOther(string mediaId, Stream stream,
    string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/material/
            get_material?access_token={0}", accessToken);
    Utils.DownloadByPost(url, JsonConvert.SerializeObject(new { media_id =
        mediaId }), stream);
}
```

3.2.4 删除永久素材

由于每个公众号可上传的数量是有限制的，因此开发者可以根据本接口删除不再使用的永久素材，以节省空间。接口地址如下：

https://api.weixin.qq.com/cgi-bin/material/del_material?access_token=ACCESS_TOKEN

HTTP 请求方式：**POST**。



调用示例: {"media_id":MEDIA_ID}。

代码清单 3-24 所示为删除素材的方法。

代码清单 3-24

```
public static ErrorEntity Del(string mediaId, string accessToken)
{
    var url="https://api.weixin.qq.com/cgi-bin/material/del_material?
access_token={0}";
    return Utils.PostResult<ErrorEntity>(new { media_id = mediaId },
    string.Format(url, accessToken));
}
```

3.2.5 修改永久图文素材

开发者可以通过本接口对永久图文素材进行修改。如果是多图文,则每次调用的时候需要指定要修改的图文项的索引。接口地址如下:

https://api.weixin.qq.com/cgi-bin/material/update_news?access_token=ACCESS_TOKEN

HTTP 请求方式: **POST**。

调用示例如代码清单 3-25 所示。

代码清单 3-25

```
{
  "media_id":MEDIA_ID,
  "index":INDEX,
  "articles":
  {
    "title": TITLE,
    "thumb_media_id": THUMB_MEDIA_ID,
    "author": AUTHOR,
    "digest": DIGEST,
    "show_cover_pic": SHOW_COVER_PIC(0 / 1),
    "content": CONTENT,
    "content_source_url": CONTENT_SOURCE_URL
  }
}
```



请求成功后，返回的 JSON 字符串格式如下：

```
{"errcode": ERRCODE, "errmsg": ERRMSG}
```

代码清单 3-26 所示为修改永久图文素材的方法。

代码清单 3-26

```
/// <summary>
/// 修改永久图文素材
/// </summary>
/// <param name="mediaId">图文素材 ID</param>
/// <param name="accessToken">调用凭据</param>
/// <param name="index">要更新的文章在图文消息中的位置（多图文消息时，此字段才有意
/// 义），第一篇为 0</param>
/// <param name="article">图文实体。此处表示的是修改后的图文信息</param>
public static ErrorEntity Update(string mediaId, string accessToken, int
index, Article article)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/material/update_
news?access_token={0}", accessToken);
    var obj = new
    {
        media_id = mediaId, index = index, articles = article
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

3.2.6 获取永久素材总数

此接口主要是做素材汇总使用的，实用性不是很大。接口地址如下：

https://api.weixin.qq.com/cgi-bin/material/get_materialcount?access_token=ACCESS_TOKEN

HTTP 请求方式：GET。

返回示例如下所示。



```
{  
    "voice_count":COUNT,  
    "video_count":COUNT,  
    "image_count":COUNT,  
    "news_count":COUNT  
}
```

根据返回的示例创建实体类 `MaterialCount`，如代码清单 3-27 所示。

代码清单 3-27

```
public class MaterialCount : ErrorEntity  
{  
    /// <summary>  
    /// 语音总数量  
    /// </summary>  
    public int voice_count { get; set; }  
    /// <summary>  
    /// 视频总数量  
    /// </summary>  
    public int video_count { get; set; }  
    /// <summary>  
    /// 图片总数量  
    /// </summary>  
    public int image_count { get; set; }  
    /// <summary>  
    /// 图文总数量  
    /// </summary>  
    public int news_count { get; set; }  
}
```

具体实现如代码清单 3-28 所示。

代码清单 3-28

```
public static MaterialCount GetCount(string accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/cgi-bin/material/get_  
materialcount?access_token={0}", accessToken);
```




```
return Utils.GetResult<MaterialCount>(url);  
}
```

3.2.7 获取永久素材列表

获取永久素材列表，也包含公众号在公众平台官网素材管理模块中新建的图文消息、语音、视频等素材。接口地址如下：

https://api.weixin.qq.com/cgi-bin/material/batchget_material?access_token=ACCESS_TOKEN

HTTP 请求方式：**POST**。

调用示例：**{"type":TYPE,"offset":OFFSET,"count":COUNT}**。

其中，type 表示的是素材的类型，包括图片（image）、视频（video）、语音（voice）、图文（news）。offset 表示开始调用的索引；0 表示从第一个素材返回；count 表示返回的素材的数量，取值在 1~20 之间。

在调用接口成功后，如果请求的类型是图文，则返回的 JSON 数据包如代码清单 3-29 所示。

代码清单 3-29

```
{  
  "total_count": TOTAL_COUNT,  
  "item_count": ITEM_COUNT,  
  "item": [{  
    "media_id": MEDIA_ID,  
    "content": {  
      "news_item": [{  
        "title": TITLE,  
        "thumb_media_id": THUMB_MEDIA_ID,  
        "show_cover_pic": SHOW_COVER_PIC(0 / 1),  
        "author": AUTHOR,  
        "digest": DIGEST,  
        "content": CONTENT,  
        "content_source_url": CONTENTN_SOURCE_URL  
      }],  
      //多图文消息会在此处有多篇文章  
    }  
  }]
```



```
    ],  
    },  
    "update_time": UPDATE_TIME  
  },  
  //可能有多条图文消息 item 结构  
]  
}
```

如果请求的是其他类型（图片、语音、视频），则返回的 JSON 数据包如代码清单 3-30 所示。

代码清单 3-30

```
{  
  "total_count": TOTAL_COUNT,  
  "item_count": ITEM_COUNT,  
  "item": [{  
    "media_id": MEDIA_ID,  
    "name": NAME,  
    "update_time": UPDATE_TIME  
  },  
  //可能会有多个素材  
]  
}
```

从上述两段代码清单中可以看出，无论请求的素材类型是什么，都有 `total_count`（该类型的素材的总数）、`item_count`（本次调用获取的素材的数量）、`item`（素材列表）属性。而 `item` 中的每一项都有 `media_id`（素材 ID）和 `update_time`（素材的最后更新时间）。唯一不同的是图文素材有 `content` 属性，而其他类型有 `name` 属性。根据 JSON 数据包，创建实体类 `MaterialList`，如代码清单 3-31 所示。

代码清单 3-31

```
using System.Collections.Generic;  
using WxApi.SendEntity;  
namespace WxApi.ReceiveEntity  
{  
    /// <summary>  
    /// 获取素材列表时，返回的实体
```



```
/// </summary>
public class MaterialList : ErrorEntity
{
    public int total_count { get; set; }
    public int item_count { get; set; }
    public List<MaterialItem> item { get; set; }
}

public class MaterialItem
{
    public string media_id { get; set; }
    /// <summary>
    /// 素材的最后更新时间
    /// </summary>
    public int update_time { get; set; }
    /// <summary>
    /// 请求类型为非图文时有效
    /// </summary>
    public string name { get; set; }
    /// <summary>
    /// 图文内容（请求类型为 news 时有效）
    /// </summary>
    public MaterialItemNews content { get; set; }
}

public class MaterialItemNews
{
    /// <summary>
    /// 多图文列表
    /// </summary>
    public List<Article> news_item { get; set; }
}
}
```

具体实现如代码清单 3-32 所示。

代码清单 3-32

```
/// <summary>
/// 获取永久素材列表，会包含公众号在公众平台官网素材管理模块中新建的图文消息、语音、视
/// 频等素材（但需要先通过获取素材列表来获知素材的 media_id）
```



```

///临时素材无法通过本接口获取
/// </summary>
/// <param name="mediaType">素材类型</param>
/// <param name="index">从全部素材的指定索引开始返回
/// 0 表示从第一个素材返回</param>
/// <param name="count">返回素材的数量，取值在 1 到 20 之间</param>
/// <param name="accessToken">全局票据</param>
public static MaterialList GetList(MaterialType mediaType, int index, int
count, string accessToken)
{
    if (count < 1 || count > 20)
    {
        return new MaterialList { ErrCode = -2,
            ErrDescription = "素材的数量，取值在 1 到 20 之间" };
    }
    if (index < 0)
    {
        return new MaterialList { ErrCode = -2,
            ErrDescription = "索引不能小于 0" };
    }
    var url =
        string.Format("
        https://api.weixin.qq.com/cgi-bin/material/batchget_material
        ?access_token={0}", accessToken);
    var obj = new { type = mediaType.ToString(), offset = index, count = count };
    return Utils.PostResult<MaterialList>(obj, url);
}

```

3.3 接收消息

当微信公众号的订阅用户向公众账号发消息时，微信服务器将 POST 消息的 XML 数据包到开发者填写的 URL 上，如文本、图片、语音、视频等消息。开发者需要根据各个消息类型的数据包的格式解析 XML，进行相应的业务处理。比如，当用户发送文本内容“天气”时，程序中解析出用户的请求，然后将天气信息回复给用户。再比如，用户发送的是地理位置，则我们可以从推送的 XML 中获取到用户的经纬度，然后根据经纬度获取附近



的商家列表，或者用户到指定门店的距离等。这种模式非常像 10086 的短信营业厅，即根据用户指令，做相应的处理。

在项目 WxApi 的 Utils 类中添加获取微信服务器 POST 的消息包的代码，如代码清单 3-33 所示。

代码清单 3-33

```
/// <summary>
/// 获取当前请求的数据包内容
/// </summary>
public static string GetRequestData()
{
    //获取传入的 HTTP 实体主体的内容
    using (var stream = HttpContext.Current.Request.InputStream)
    {
        using (var reader = new StreamReader(stream, Encoding.UTF8))
        {
            return reader.ReadToEnd();
        }
    }
}
```

3.3.1 普通消息实体映射

微信中的消息大致可以分为两种：普通消息和事件消息。普通消息包括文本、图片、语音、视频、地理位置及链接。在第 2 章中讲到，微信公众号的工作方式是基于“请求”、“响应”的方式，所以每次用户发送给公众号的消息都相当于一次请求，而服务器必须进行响应。**微信服务器在 5 秒内收不到响应就会断掉连接，并且重新发起请求，总共重试 3 次。假如服务器无法保证在 5 秒内处理并回复，则可以直接回复空串，微信服务器不会对此做任何处理，并且不会发起重试。这里说的空串，指的是空字符串，即 Response.Write(" ")。**

虽然消息被分了类，但各个类型消息的数据包还是有相同的地方，这样有利于我们将其进行抽象。下面就使用代码清单 3-31 的方法来获取各个消息的数据包。首先，在 2.2.4 节中讲到的 wx.ashx 页面中，找到判断 HTTP 数据传输方式的代码，在 else 中添加代码清单 3-34 中的代码。

代码清单 3-34

```
//微信服务器消息处理业务逻辑
var xml = Utils.GetRequestData();
```




```
context.Response.Write("");
```

然后在微信中发送文本消息“hello world”给公众号，如图 3-5 所示。



图 3-5

代码清单 3-35 则是接收到的消息内容。

代码清单 3-35

```
<xml>
<ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
<FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
<CreateTime>1423152010</CreateTime>
<MsgType><![CDATA[text]]></MsgType>
<Content><![CDATA[hello world]]></Content>
<MsgId>6112391340390100796</MsgId>
</xml>
```

Content 节点表示的就是用户发送的消息内容。

当发送到的消息内容是图片消息时，接收到的消息内容如代码清单 3-36 所示。

代码清单 3-36

```
<xml>
<ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
<FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
<CreateTime>1423153766</CreateTime>
<MsgType><![CDATA[image]]></MsgType>
<PicUrl><![CDATA[http://mmbiz.qpic.cn/mmbiz/jke86QtK7tjTs3cehU9F5gLUe1
znmRwI2DicpM8GsNda3ZjTgJ2Edb4YevPk2SMMIffj8fdvsMVTiczKPbUMgSFA/0]]>
</PicUrl>
```



```
<MsgId>6112398882352672959</MsgId>
<MediaId><![CDATA[1H8DpxxwZM0YqVkte0WFpwZXw_c8HPA3HkQtKzbldHOciOv
PpA541vpUcy9sWAOS]]></MediaId>
</xml>
```

在上述代码中，PicUrl 表示的是发送的图片的 URL，此链接可以直接在浏览器中打开查看。MediaId 是图片消息的媒体 ID，可以调用获取临时素材接口获取数据。

语音代码的数据包格式如代码清单 3-37 所示。

代码清单 3-37

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1423154312</CreateTime>
  <MsgType><![CDATA[voice]]></MsgType>
  <MediaId><![CDATA[jHzhR0mBQ_Tv3rhqxV4HhBNnqUhX88-bH9jNwz6s6K7IDaVvha
81YisQORqYsWSS]]></MediaId>
  <Format><![CDATA[amr]]></Format>
  <MsgId>6112401227201380352</MsgId>
  <Recognition><![CDATA[你好]]></Recognition>
</xml>
```

语音的数据包和图片的数据包大致相同。需要注意的是，用户每次发送语音给开通语音识别功能的公众号时，微信会在推送的 XML 数据包中，相对于未开通此功能的数据包增加一个 Recognition 字段，这个字段表示的是语音识别的结果。

代码清单 3-38 是发送视频消息时，接收到的 XML 数据包，其中 ThumbMediaId 表示视频消息缩略图的媒体 id，也可以使用获取临时素材接口获取数据。

代码清单 3-38

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1423154880</CreateTime>
  <MsgType><![CDATA[video]]></MsgType>
  <MediaId><![CDATA[ubmQzRbsGsxccBzpJE3RB7pww16rAQYKtAIR3FKZ1G13QKTRGG
W2LuCMrcWE1Cg4]]></MediaId>
```



```
<ThumbMediaId><![CDATA[ubmQzRbsGsxccBzpJE3RB7pwwl6rAQYKtaIR3FKZlG13Q
KTRGGW2LuCMrcWE1Cg4]]></ThumbMediaId>
<MsgId>6112403666946240938</MsgId>
</xml>
```

代码清单 3-39 是发送地理位置消息时，接收到的 XML 数据包。其中，Location_X 表示的是地理位置的纬度，Location_Y 则表示的是经度，Scale 是地图的缩放大小，Label 是地理位置信息的描述。地理位置定位是微信中用得比较多的，后期将结合实际案例讲述此功能的用法。

代码清单 3-39

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXIttyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1423156291</CreateTime>
  <MsgType><![CDATA[location]]></MsgType>
  <Location_X>30.528843</Location_X>
  <Location_Y>114.347305</Location_Y>
  <Scale>16</Scale>
  <Label><![CDATA[武汉市武昌区武珞路 675 号]]></Label>
  <MsgId>6112409727145095809</MsgId>
</xml>
```

代码清单 3-40 是发送链接消息时，接收到的 XML 数据包。其中，Title 是链接的标题，Description 是链接的描述，Url 是链接的地址。

代码清单 3-40

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXIttyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1423156830</CreateTime>
  <MsgType><![CDATA[link]]></MsgType>
  <Title><![CDATA[快过年了，给爸妈的礼物你准备好了吗?]]></Title>
  <Description><![CDATA[转眼又到了一年团圆的日子，是该放下所有的劳累与困顿，带一份健康与安心回家~]]></Description>
  <Url><![CDATA[http://mp.weixin.qq.com/s?__biz=MjM5MTgyNzQ5Nw==&mid=
203830787&idx=1&sn=d9ab438b85f3684dd3b053a38a96bc0b#rd]]></Url>
```



```
<MsgId>6112412042132468411</MsgId>  
</xml>
```

可能很多人不知道怎么给公众号发送链接，发送消息给公众号的时候也没有选择链接的入口。其实，发送链接的入口在“我的收藏”中，如图 3-6 所示。单击“我的收藏”，选择收藏夹中的链接即可。



图 3-6

也许聪明绝顶的你应该发现了，所有消息的数据包中（包括事件消息）都包含表 3-2 中的几个字段。

表 3-2

参 数	说 明
ToUserName	接收方微信号
FromUserName	发送方微信号。若为普通用户，则是一个 openid
CreateTime	消息创建时间
MsgType	消息类型



既然所有的消息体都有上面的几个字段，那就可以将消息抽象出一个基类，之后不同的消息实体继承这个基类即可。下面是具体的实现过程。

首先，在 WxApi 项目中，新建文件夹，用来存放和消息相关的实体。新建抽象类 BaseMsg，如代码清单 3-41 所示。

代码清单 3-41

```
/// <summary>
/// 所有消息类型的基类
/// </summary>
public abstract class BaseMsg
{
    /// <summary>
    /// 开发者微信号
    /// </summary>
    public string ToUserName { get; set; }
    /// <summary>
    /// 发送方账号（一个 openid）
    /// </summary>
    public string FromUserName { get; set; }
    /// <summary>
    /// 消息创建时间（整型）
    /// </summary>
    public string CreateTime { get; set; }
    /// <summary>
    /// 消息类型
    /// </summary>
    public MsgType MsgType { get; set; }
}
```

在基类中，属性 MsgType 的类型是枚举 MsgType。消息的类型在本节开头已经讲了，为了方便管理和代码编写，我们把消息类型定义成枚举 MsgType，如代码清单 3-42 所示。

代码清单 3-42

```
/// <summary>
/// 消息类型枚举
```




```
/// </summary>
public enum MsgType
{
    /// <summary>
    /// 文本类型
    /// </summary>
    TEXT,
    /// <summary>
    /// 图片类型
    /// </summary>
    IMAGE,
    /// <summary>
    /// 语音类型
    /// </summary>
    VOICE,
    /// <summary>
    /// 视频类型
    /// </summary>
    VIDEO,
    /// <summary>
    /// 地理位置类型
    /// </summary>
    LOCATION,
    /// <summary>
    /// 链接类型
    /// </summary>
    LINK,
    /// <summary>
    /// 事件类型
    /// </summary>
    EVENT
}
```

这里说明一下：C#中的 `event` 是关键字，所以 `event` 在枚举中就不能使用了。因此，为了统一，这里的枚举全部使用大写的。

基类创建好后，根据 XML 数据包，创建各个消息类型的实体，并继承基类 `BaseMsg`



(见代码清单 3-43 至代码清单 3-48)。

代码清单 3-43 (文本消息实体)

```
/// <summary>
/// 文本实体
/// </summary>
public class TextMsg : BaseMsg
{
    /// <summary>
    /// 消息内容
    /// </summary>
    public string Content { get; set; }
    /// <summary>
    /// 消息 ID, 64 位整型
    /// </summary>
    public string MsgId { get; set; }
}
```

代码清单 3-44 (语音消息实体)

```
public class VoiceMsg : BaseMsg
{
    /// <summary>
    /// 缩略图 ID
    /// </summary>
    public string MsgId { get; set; }
    /// <summary>
    /// 格式
    /// </summary>
    public string Format { get; set; }
    /// <summary>
    /// 媒体 ID
    /// </summary>
    public string MediaId { get; set; }
    /// <summary>
    /// 语音识别结果
    /// </summary>
}
```



```
public string Recognition { get; set; }  
}
```

代码清单 3-45 (视频消息实体)

```
public class VideoMsg : BaseMsg  
{  
    /// <summary>  
    /// 缩略图 ID  
    /// </summary>  
    public string ThumbMediaId { get; set; }  
    /// <summary>  
    /// 消息 ID, 64 位整型  
    /// </summary>  
    public string MsgId { get; set; }  
    /// <summary>  
    /// 媒体 ID  
    /// </summary>  
    public string MediaId { get; set; }  
}
```

代码清单 3-46 (链接消息实体)

```
public class LinkMsg : BaseMsg  
{  
    /// <summary>  
    /// 缩略图 ID  
    /// </summary>  
    public string MsgId { get; set; }  
    /// <summary>  
    /// 标题  
    /// </summary>  
    public string Title { get; set; }  
    /// <summary>  
    /// 描述  
    /// </summary>  
    public string Description { get; set; }  
    /// <summary>  
    /// 链接地址
```



```
/// </summary>
public string Url { get; set; }
}
```

代码清单 3-47 (图片消息实体)

```
public class ImgMsg : BaseMsg
{
    /// <summary>
    /// 图片路径
    /// </summary>
    public string PicUrl { get; set; }
    /// <summary>
    /// 消息 ID, 64 位整型
    /// </summary>
    public string MsgId { get; set; }
    /// <summary>
    /// 媒体 ID
    /// </summary>
    public string MediaId { get; set; }
}
```

代码清单 3-48 (地理位置消息实体)

```
public class LocationMsg:BaseMsg
{
    /// <summary>
    /// 地理位置维度
    /// </summary>
    public string Location_X { get; set; }
    /// <summary>
    /// 地理位置经度
    /// </summary>
    public string Location_Y { get; set; }
    /// <summary>
    /// 地图缩放大小
    /// </summary>
    public int Scale { get; set; }
    /// <summary>
```




```
/// 地理位置信息
/// </summary>
public string Label { get; set; }
/// <summary>
/// 链接地址
/// </summary>
public string Url { get; set; }
/// <summary>
/// 消息 ID
/// </summary>
public string MsgId { get; set; }
}
```

3.3.2 事件消息实体映射

事件消息包括：

- 关注/取消关注事件；
- 扫描带参数二维码事件；
- 上报地理位置事件；
- 自定义菜单事件。

其中，自定义菜单事件包含一组事件，详情将在后面的章节一一介绍。想必聪明的你看到这里应该知道怎么做了吧。按照常规做法，我们应该将这些项定义成枚举类型，如代码清单 3-49 所示。

代码清单 3-49

```
/// <summary>
/// 事件类型枚举
/// </summary>
public enum EventType
{
    /// <summary>
    /// 非事件类型

```




```
/// </summary>
NOEVENT,
/// <summary>
/// 订阅
/// </summary>
SUBSCRIBE,
/// <summary>
/// 取消订阅
/// </summary>
UNSUBSCRIBE,
/// <summary>
/// 扫描带参数的二维码
/// </summary>
SCAN,
/// <summary>
/// 地理位置
/// </summary>
LOCATION,
/// <summary>
/// 单击按钮
/// </summary>
CLICK,
/// <summary>
/// 链接按钮
/// </summary>
VIEW,
/// <summary>
/// 扫码推事件
/// </summary>
SCANCODE_PUSH,
/// <summary>
/// 扫码推事件且弹出“消息接收中”提示框
/// </summary>
SCANCODE_WAITMSG,
/// <summary>
/// 弹出系统拍照发图
/// </summary>
```



```
PIC_SYSPHOTO,  
/// <summary>  
/// 弹出拍照或者相册发图  
/// </summary>  
PIC_PHOTO_OR_ALBUM,  
/// <summary>  
/// 弹出微信相册发图器  
/// </summary>  
PIC_WEIXIN,  
/// <summary>  
/// 弹出地理位置选择器  
/// </summary>  
LOCATION_SELECT,  
/// <summary>  
/// 模板消息推送  
/// </summary>  
TEMPLATESENDJOBFINISH
```

```
}
```

这里列举出了大部分消息类型，由于微信的接口更新比较频繁，如果后期有新增的事件类型，则需在此枚举类型中增加对应的项。

与 3.3.1 节一样，在定义消息实体映射前，我们需要知道各个消息类型的数据包格式。下面是关注/取消关注事件与上报地理位置事件的数据包格式（自定义菜单事件和扫描二维码事件将在后面的章节详细讲解），如代码清单 3-50 和代码清单 3-51 所示。

代码清单 3-50（关注/取消关注事件）

```
<xml>  
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>  
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>  
  <CreateTime>1423291412</CreateTime>  
  <MsgType><![CDATA[event]]></MsgType>  
  <Event><![CDATA[unsubscribe]]></Event>  
  <EventKey><![CDATA[]]></EventKey>  
</xml>
```

代码清单 3-51（上报地理位置事件）

```
<xml>
```



```
<ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
<FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
<CreateTime>1423294068</CreateTime>
<MsgType><![CDATA[event]]></MsgType>
<Event><![CDATA[LOCATION]]></Event>
<Latitude>30.528910</Latitude>
<Longitude>114.355293</Longitude>
<Precision>30.000000</Precision>
</xml>
```

在代码清单 3-50 中, Event 的值为 subscribe 时, 表示订阅事件; 为 unsubscribe 表示取消订阅事件。所以, 此代码是取消订阅事件的 XML 数据包。

在代码清单 3-51 中, Latitude 表示的是地理位置纬度, Longitude 表示的是地理位置经度, Precision 表示的是地理位置精度。需要注意的是, 此消息和普通消息中的地理位置消息不同, 普通消息中的地理位置是用户手动发送给公众号的; 而上报地理位置接口则是拥有此接口权限的公众号开启后, 在征得用户同意的情况下, 由微信客户端自动获取, 并提交给开发者的服务器的。在公众平台后台“开发者中心”找到如图 3-7 所示的权限表, 单击“开启”按钮, 弹出如图 3-8 所示的提示窗口, 选择获取地理位置的模式即可开启。

生成带参数二维码	
获取用户地理位置	开启
获取用户基本信息	
获取关注者列表	
用户分组接口	

图 3-7

开启成功后, 用户在进入对话窗口时, 微信客户端会询问用户是否允许公众号使用地理位置, 如图 3-9 所示。用户同意后, 如果在开启此功能的时候选择的是“用户进行对话时上报一次”, 则微信客户端在此次会话中, 只上报一次。如果选择的是“用户进行对话后每隔 5s 上报一次”, 则每隔 5 秒就会上报一次地理位置。用户不同意, 则不上报。

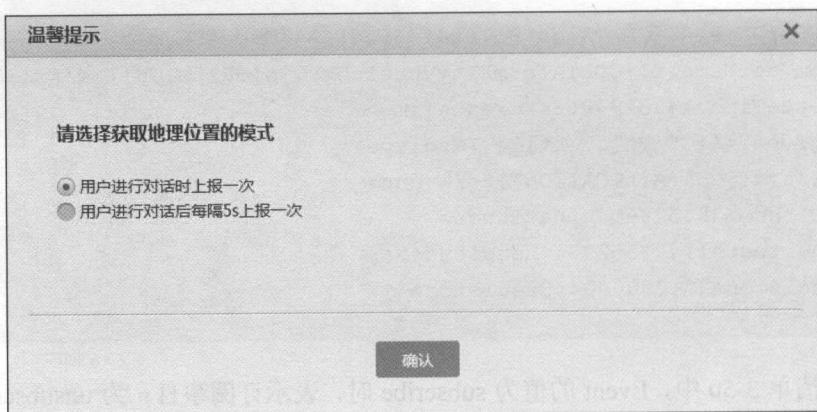


图 3-8

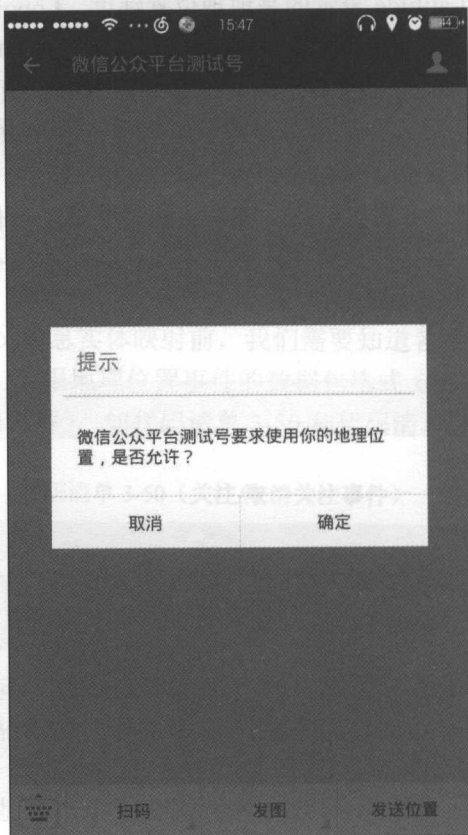


图 3-9



事件类型的消息除了和普通消息一样都拥有 ToUserName、FromUserName 等字段，还有 Event 字段，Event 表示的是事件的类型。所以，我们可以抽象出一个事件基类，如代码清单 3-52 所示。

代码清单 3-52

```
public class EventMsg : BaseMsg
{
    public EventType Event { get; set; }
}
```

最后根据 XML 数据包创建事件消息的实体，如代码清单 3-53 所示。

代码清单 3-53 (订阅/取消订阅事件)

```
public class SubEventMsg : EventMsg
{
}
```

基类中已经包括了常规订阅事件中所有的属性，所以在 SubEventMsg 类中，并没有添加任何的属性或字段（见代码清单 3-54）。那既然有常规订阅事件，也就意味着有非常规订阅事件，也就是后面章节中将会讲述的带参数二维码接口（见代码清单 3-129）。

代码清单 3-54 (上报地理位置)

```
public class LocationEventMsg : EventMsg
{
    /// <summary>
    /// 地理位置纬度
    /// </summary>
    public string Latitude { get; set; }
    /// <summary>
    /// 地理位置经度
    /// </summary>
    public string Longitude { get; set; }
    /// <summary>
    /// 地理位置精度
    /// </summary>
    public string Precision { get; set; }
}
```




3.3.3 消息数据包解析

消息实体创建完成后，我们需要将接收的 XML 数据包转换成对象，以便于处理程序的逻辑。从消息实体中也可以看出，每个消息实体都拥有相同基类的属性，为了节约编码量及提高代码的可阅读性，可以利用反射，先给共有的属性赋值，然后再遍历未处理的属性，给各个消息实体独有的属性赋值。代码清单 3-55 中方法 ConvertObj 的功能是将微信 POST 过来的 XML 数据包转换成对应的实体对象。

代码清单 3-55

```
public static T ConvertObj<T>(string xmlstr)
{
    try
    {
        XElement xdoc = XElement.Parse(xmlstr);
        //获取转换的数据类型
        var type = typeof(T);
        //创建实例
        var t = Activator.CreateInstance<T>();
        #region 基础属性赋值
        var ToUserName = type.GetProperty("ToUserName");
        ToUserName.SetValue(t,
            Convert.ChangeType(xdoc.Element("ToUserName").Value,
                ToUserName.PropertyType), null);
        xdoc.Element("ToUserName").Remove();

        var FromUserName = type.GetProperty("FromUserName");
        FromUserName.SetValue(t,
            Convert.ChangeType(xdoc.Element("FromUserName").Value,
                FromUserName.PropertyType), null);
        xdoc.Element("FromUserName").Remove();

        var CreateTime = type.GetProperty("CreateTime");
        CreateTime.SetValue(t,
            Convert.ChangeType(xdoc.Element("CreateTime").Value,
                CreateTime.PropertyType), null);
        xdoc.Element("CreateTime").Remove();

        var MsgType = type.GetProperty("MsgType");
```



```
string msgtype = xdoc.Element("MsgType").Value.ToUpper();
MsgType.SetValue(t, (MsgType)Enum.Parse(typeof(MsgType), msgtype),
null);
xdoc.Element("MsgType").Remove();

//判断消息类型是否是事件
if (msgtype == "EVENT")
{
    //获取事件类型
    var EventType = type.GetProperty("Event");
    string eventtype = xdoc.Element("Event").Value.ToUpper();
    EventType.SetValue(t, (EventType)Enum.Parse(typeof(EventType),
eventtype), null);
    xdoc.Element("Event").Remove();
}
#endregion
//遍历 XML 节点
foreach (XElement element in xdoc.Elements())
{
    //根据 XML 节点的名称, 获取实体的属性
    var pr = type.GetProperty(element.Name.ToString());
    //给属性赋值
    pr.SetValue(t,
    Convert.ChangeType(element.Value, pr.PropertyType), null);
}
return t;
}
catch (Exception)
{
    return default(T);
}
}
```

有了将 XML 转换给对象的方法, 下一步就是创建一个类, 用于根据 XML 获得实体对象。并且前面章节中也提到了, 微信服务器在 5 秒内收不到响应就会断掉连接, 并且重新发起请求, 总共重试 3 次。这样的话, 问题就来了。有这样一个场景: 当用户关注微信账号时, 获取当前用户信息, 然后将信息写到数据库中。类似于 PC 端网站的注册。可能由于在这个关注事件中, 我们需要处理的业务逻辑比较复杂, 如送积分、写用户日志、分配用户组, 等等……一系列的逻辑需要执行, 或者网络环境比较复杂, 无法保证 5 秒内响应



当前用户的操作。如果当操作尚未完成时，微信服务器又给服务器推送了一条相同的关注事件，我们将再次执行那些逻辑，这样就有可能导致数据库中出现重复的数据。解决问题的方法很简单，我们只需要在处理业务逻辑之前，将用户的前 n 条消息缓存起来，每次有新消息推送时，先将已经失效的消息移除（5 秒重试一次，总共重试 3 次，也就是 15 秒，为保险起见可移除 20 秒之前的消息），然后去缓存中查询是否已经存在此消息。如果不存在，则将此消息也加入缓存；否则直接 `return null`，不做任何处理。普通消息的唯一性是使用 `MsgId` 判断的，而事件消息则使用 `FromUserName + CreateTime`。下面是具体的实现步骤。

首先，新建一个类 `BMsg`，如代码清单 3-56 所示

代码清单 3-56

```
public class BMsg
{
    /// <summary>
    /// 发送者标识
    /// </summary>
    public string FromUser { get; set; }
    /// <summary>
    /// 消息表示。普通消息时，为 MsgId；事件消息时，为事件的创建时间
    /// </summary>
    public string MsgFlag { get; set; }
    /// <summary>
    /// 添加到队列的时间
    /// </summary>
    public DateTime CreateTime { get; set; }
}
```

然后，新建类 `MsgFactory`，类中有一个私有的静态变量 `_queue`，用来缓存消息列表；一个静态方法 `LoadMsg`，此方法接收一个 `List<MsgHandlerEntity>` 类型的参数 `mheList`，`mheList` 表示的是各个消息类型的处理程序列表。`MsgHandlerEntity` 的代码如代码清单 3-57 所示。

代码清单 3-57

```
public class MsgHandlerEntity
{

```



```
/// <summary>
/// 消息类型
/// </summary>
public MessageType MessageType { get; set; }
/// <summary>
/// 事件类型
/// </summary>
public EventType EventType { get; set; }
public Action<BaseMsg> Action { get; set; }
/// <summary>
/// 消息处理程序列表
/// </summary>
public static List<MsgHandlerEntity> MsgHandlerEntities;
}
```

这样设计的好处是，当我们在调用 LoadMsg 方法时，只需将各个类型消息的处理程序实例化成对象，然后添加到列表中，LoadMsg 的方法体中根据具体的消息类型或事件类型绑定对应的回调程序，并执行，如代码清单 3-58 所示。

代码清单 3-58

```
public class MsgFactory
{
    private static List<BMsg> _queue;
    /// <summary>
    /// 加载消息，并绑定各消息类型的处理程序
    /// </summary>
    /// <param name="mheList">消息处理程序列表</param>
    public static void LoadMsg(List<MsgHandlerEntity> mheList)
    {
        //判断队列是否为空，为空则实例化
        if (_queue == null)
        {
            _queue = new List<BMsg>();
        }
        else
        {
            //保留 20 秒内未响应的消息
            _queue = _queue.Where(q => q.CreateTime.AddSeconds(20) >
```




```
DateTime.Now).ToList();
    }
    //获取数据包
    string postStr = Utils.GetRequestData();
    XElement xdoc = XElement.Parse(postStr);
    var msgtype = xdoc.Element("MsgType").Value.ToUpper();
    var FromUserName = xdoc.Element("FromUserName").Value;
    var CreateTime = xdoc.Element("CreateTime").Value;
    //获取消息类型
    MsgType type = (MsgType)Enum.Parse(typeof(MsgType), msgtype);
    //如果不是事件类型
    if (type != MsgType.EVENT)
    {
        //判断队列中是否已经存在此消息。如果不存在，则将此消息加入队列中；否则返回 null
        var MsgId = xdoc.Element("MsgId").Value;
        if (_queue.FirstOrDefault(m => m.MsgFlag == MsgId) == null)
        {
            _queue.Add(new BMsg
            {
                CreateTime = DateTime.Now,
                FromUser = FromUserName,
                MsgFlag = MsgId
            });
        }
        else
        {
            return;
        }
    }
    else
    {
        //判断队列中是否已经存在此消息。如果不存在，则将此消息加入队列中；否则返回 null
        if (_queue.FirstOrDefault(m => m.MsgFlag == CreateTime && m.FromUser == FromUserName) == null)
        {
            _queue.Add(new BMsg
            {
                CreateTime = DateTime.Now,
```




```
        FromUser = FromUserName,
        MsgFlag = CreateTime
    });
}
else
{
    return;
}
}
//消息实体对象
BaseMsg msg = null;
//事件类型。默认为 NOEVENT, 即非事件类型
EventType eventtype = EventType.NOEVENT;
switch (type)
{
    //获取文本消息实体
    case MsgType.TEXT:
        msg = Utils.ConvertObj<TextMsg>(postStr);
        break;
    //获取图片消息实体
    case MsgType.IMAGE: msg = Utils.ConvertObj<ImgMsg>(postStr); break;
    //获取视频消息实体
    case MsgType.VIDEO: msg = Utils.ConvertObj<VideoMsg>(postStr);
        break;
    //获取音频消息实体
    case MsgType.VOICE: msg = Utils.ConvertObj<VoiceMsg>(postStr);
        break;
    //获取链接消息实体
    case MsgType.LINK: msg = Utils.ConvertObj<LinkMsg>(postStr); break;
    //获取地理位置消息实体
    case MsgType.LOCATION: msg = Utils.ConvertObj<LocationMsg>(postStr);
        break;
    //获取事件消息实体
    case MsgType.EVENT://事件类型
    {
        eventtype = (EventType)Enum.Parse(typeof(EventType),
        xdoc.Element("Event").Value.ToUpper());
        switch (eventtype)
```



```
{
    //订阅与取消订阅事件
    case EventType.UNSUBSCRIBE: msg =
Utils.ConvertObj<SubEventMsg>(postStr); break;
    //上报地理位置事件
    case EventType.LOCATION: msg =
Utils.ConvertObj<LocationEventMsg>(postStr); break;
    default:
        msg = Utils.ConvertObj<EventMsg>(postStr); break;
}
} break;
default:
    msg = Utils.ConvertObj<BaseMsg>(postStr); break;

}
//根据消息类型, 获取回调程序
var ac = GetAction(mheList, type, eventType);
//如果回调程序存在, 并且消息实体转换成功, 则执行回调程序
if (ac != null && msg != null)
{
    ac(msg);
}
}
/// <summary>
/// 根据消息类型, 获取回调程序
/// </summary>
/// <param name="actions">回调委托列表</param>
/// <param name="msgType">消息类型</param>
/// <returns>回调委托</returns>
private static Action<BaseMsg> GetAction(List<MsgHandlerEntity>
mheList, MsgType msgType, EventType eventType)
{
    MsgHandlerEntity temp = mheList.FirstOrDefault(mhe =>
    {
        if (msgType != MsgType.EVENT)
            return mhe.MsgType == msgType;
        return mhe.EventType == eventType;
    });
}
```



```
return temp.Action;
```

```
}
```

```
}
```

到这里为止，处理接收消息的功能基本实现。在后面章节中讲到自定义菜单事件、扫描二维码事件以及消息体签名加密时，这里的代码只需简单调整一下，即可实现业务逻辑。

3.3.4 消息处理页面示例

3.3.3 节已经将处理 XML 数据的业务逻辑封装好了，本节就让我们一起来看看怎么调用吧。在 2.2.4 节中，我们讲到，如果请求类型为 POST 时，则说明是接入成功后，微信推送过来的消息，我们只需要实现此部分的业务逻辑即可。代码清单 3-59 就是 wx.ashx 的完整代码。

代码清单 3-59

```
using System.Collections.Generic;
using System.Web;
using WxApi;
using WxApi.MsgEntity;

namespace WxTest
{
    /// <summary>
    /// wx 的摘要说明
    /// </summary>
    public class wx : IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            //var ip = context.Request.UserHostAddress;
            //var ipEntity = BaseServices.GetIpArray("你的 access_token");
            //if (ipEntity!=null&&!ipEntity.ip_list.Contains(ip))
            //{
            //    context.Response.Write("非法请求");
            //    return;
            //}
            if (context.Request.HttpMethod == "GET")
            {

```



```
BaseServices.ValidUrl("qqq");
}
else
{
    // 判断 MsgHandlerEntities 是否为空。如果是，则实例化，并将各个消息类
    // 型的处理程序实体添加到此列表中
    // 因为 MsgHandlerEntities 是静态变量，只需绑定一次即可。免去了重复绑
    // 定的性能损耗
    if (MsgHandlerEntity.MsgHandlerEntities == null)
    {
        MsgHandlerEntity.MsgHandlerEntities =
new List<MsgHandlerEntity>();

        #region 文本消息处理绑定
        MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
        {
            MsgType = MsgType.TEXT,
            Action = TextHandler
        });
        #endregion

        #region 图片消息处理绑定
        MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
        {
            MsgType = MsgType.IMAGE,
            Action = ImgHandler
        });
        #endregion

        #region 语音消息处理绑定
        MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
        {
            MsgType = MsgType.VOICE,
            Action = VoiceHandler
        });
        #endregion
    }
}
```




```
#region 视频消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
{
    MsgType = MsgType.VIDEO,
    Action = VideoHandler
});
#endregion

#region 地理位置消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
{
    MsgType = MsgType.LOCATION,
    Action = LoctionHandler
});
#endregion

#region 链接消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new
MsgHandlerEntity
{
    MsgType = MsgType.LINK,
    Action = linkHandler
});
#endregion
}
MsgFactory.LoadMsg(MsgHandlerEntity.MsgHandlerEntities);
}

#region 文本消息处理程序
/// <summary>
/// 文本消息处理程序
/// </summary>
private void TextHandler(BaseMsg baseMsg)
{
    var msg = (TextMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.Content);
}
```




```
#endregion

#region 图片消息处理程序
/// <summary>
/// 图片消息处理程序
/// </summary>
private void ImgHandler(BaseMsg baseMsg)
{
    var msg = (ImgMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.PicUrl);
}
#endregion

#region 视频消息处理程序
/// <summary>
/// 视频消息处理程序
/// </summary>
private void VideoHandler(BaseMsg baseMsg)
{
    var msg = (VideoMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.ThumbMediaId);
}
#endregion

#region 语音消息处理程序
/// <summary>
/// 语音消息处理程序
/// </summary>
private void VoiceHandler(BaseMsg baseMsg)
{
    var msg = (VoiceMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.MediaId);
}
#endregion

#region 链接消息处理程序
/// <summary>
```



```
/// 链接消息处理程序
/// </summary>
private void linkHandler(BaseMsg baseMsg)
{
    var msg = (LinkMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.Description);
}
```

```
#endregion
```

```
#region 地理位置消息处理程序
```

```
/// <summary>
/// 地理位置消息处理程序
/// </summary>
private void LocationHandler(BaseMsg baseMsg)
{
    var msg = (LocationMsg)baseMsg;
    Utils.WriteTxt("/debug.txt", msg.Label);
}
```

```
#endregion
```

```
public bool IsReusable
```

```
{
    get
    {
        return false;
    }
}
```

```
}
```

```
}
```

在上述代码清单中,xxxHandler 方法中的 Utils.WriteTxt("/debug.txt", xxx)代码供调试使用,详细代码如下所示:

```
public static void WriteTxt(string path, string txt)
{
    using (FileStream fs = new
        FileStream(HttpContext.Current.Request.MapPath(path),
        FileMode.Append, FileAccess.Write))
    {
```



```
using (StreamWriter sw = new StreamWriter(fs))
{
    sw.WriteLine(txt);
    sw.Flush();
}
}
```

当用户发送消息给公众号时，程序处理后，将相应的信息写入 debug.txt 文件，如图 3-10 和图 3-11 所示。

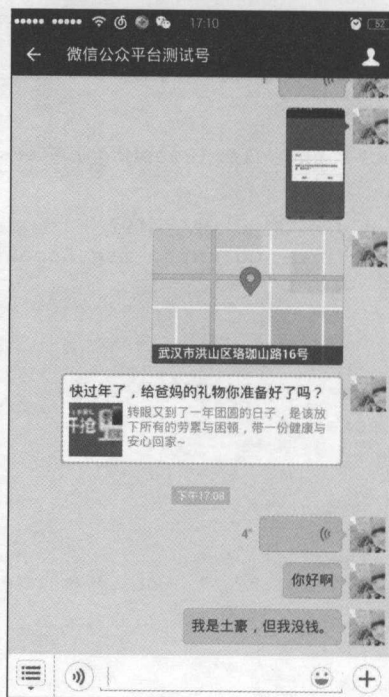


图 3-10

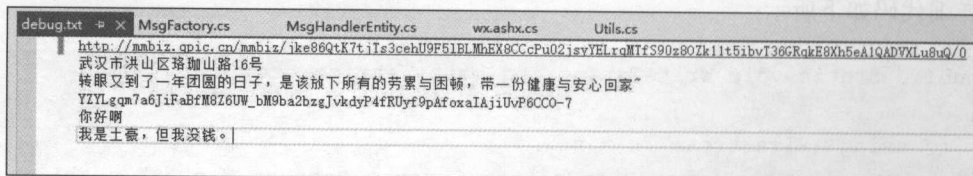


图 3-11



3.4 被动响应消息与客服接口

3.4.1 被动响应消息

在 3.3 节中讲到, 微信服务器在发出请求后, 在 5 秒内收不到响应就会断掉连接, 并且重新发起请求。所以, 我们需要及时响应微信服务器的请求。如果开发者在 5 秒内未回复任何内容, 或者开发者回复了异常数据, 则微信服务器会向当前发送消息的用户下发系统提示“该公众号暂时无法提供服务, 请稍后再试”。假如服务器无法保证在 5 秒内处理并回复, 则必须直接回复空串。微信服务器不会再对此条消息做任何处理, 并且不会发起重试。这种情况下, 可以使用客服消息接口进行异步回复。

被动响应的消息是以 XML 数据包的形式回复给用户的, 可回复的消息类型包括文本、图片、图文、语音、视频、音乐。这里需要注意的是, 回复图片等多媒体消息时, 需要预先上传素材到微信服务器。并且只有通过认证的公众号才有调用素材管理接口的权限。换句话说, 未认证的公众号可回复的消息类型只有文本和图文。

各消息类型需要的 XML 数据包结构如代码清单 3-60 到代码清单 3-65 所示。

代码清单 3-60 (回复文本消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[你好]]></Content>
</xml>
```

代码清单 3-61 (回复图片消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[image]]></MsgType>
  <Image>
  <MediaId><![CDATA[media_id]]></MediaId>
```




```
</Image>
</xml>
```

代码清单 3-62 (回复语音消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[voice]]></MsgType>
  <Voice>
  <MediaId><![CDATA[media_id]]></MediaId>
  </Voice>
</xml>
```

代码清单 3-63 (回复视频消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[video]]></MsgType>
  <Video>
  <MediaId><![CDATA[media_id]]></MediaId>
  <Title><![CDATA[title]]></Title>
  <Description><![CDATA[description]]></Description>
  </Video>
</xml>
```

代码清单 3-64 (回复音乐消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[music]]></MsgType>
  <Music>
  <Title><![CDATA[TITLE]]></Title>
  <Description><![CDATA[DESCRIPTION]]></Description>
```




```
<MusicUrl><![CDATA[MUSIC_Url]]></MusicUrl>
<HQMusicUrl><![CDATA[HQ_MUSIC_Url]]></HQMusicUrl>
<ThumbMediaId><![CDATA[media_id]]></ThumbMediaId>
</Music>
</xml>
```

代码清单 3-65 (回复图文消息)

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>12345678</CreateTime>
  <MsgType><![CDATA[news]]></MsgType>
  <ArticleCount>2</ArticleCount>
  <Articles>
    <item>
      <Title><![CDATA[title1]]></Title>
      <Description><![CDATA[description1]]></Description>
      <PicUrl><![CDATA[picurl]]></PicUrl>
      <Url><![CDATA[url]]></Url>
    </item>
    <item>
      <Title><![CDATA[title]]></Title>
      <Description><![CDATA[description]]></Description>
      <PicUrl><![CDATA[picurl]]></PicUrl>
      <Url><![CDATA[url]]></Url>
    </item>
  </Articles>
</xml>
```

各个消息类型的数据包中都包含如表 3-3 所示的字段。

表 3-3

参 数	是 否 必 需	说 明
ToUserName	是	接收方账号 (收到的 openid, 即接收消息数据包中的 FromUserName)
FromUserName	是	开发者微信号, 即接收消息数据包中的 ToUserName
CreateTime	是	消息创建时间 (整型), 此字段表示的是 UNIX 时间
MsgType	是	消息类型



文本消息的数据包中的 `Content` 字段表示的是回复的消息内容。支持换行符 `\r\n`，以及 `a` 标签。

知道了消息的数据包格式，那就可以给用户回复消息了。还记得在 3.3 节中创建的消息基类 `BaseMsg` 吧，所有的消息实体都继承这个基类，所以，我们只需在 `BaseMsg` 中添加回复消息的方法即可。代码清单 3-66 就是回复文本的代码。

代码清单 3-66

```
public virtual void ResText(string content)
{
    var resxml = new StringBuilder();
    resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",
        FromUserName);
    resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",
        ToUserName);
    resxml.AppendFormat("<CreateTime>{0}</CreateTime>",
        Utils.ConvertDateTimeInt(DateTime.Now));
    resxml.Append("<MsgType><![CDATA[text]]></MsgType>");
    resxml.AppendFormat("<Content><![CDATA[{0}]]></Content></xml>", content);
    HttpContext.Current.Response.Write(resxml);
    return;
}
```

在上述代码中，代码段 `Utils.ConvertDateTimeInt(DateTime.Now)` 的功能是将当前时间转换成 UNIX 时间戳，具体代码如下所示：

```
public static int ConvertDateTimeInt(System.DateTime time)
{
    var startTime = TimeZone.CurrentTimeZone.
        ToLocalTime(new System.DateTime(1970, 1, 1));
    return (int)(time - startTime).TotalSeconds;
}
```

定义好方法后，我们只需将要回复的文本内容传过来即可，如代码清单 3-67 所示（见图 3-12）。

代码清单 3-67

```
/// <summary>
```



```

/// 文本消息处理程序
/// </summary>
private void TextHandler(BaseMsg baseMsg)
{
    var msg = (TextMsg)baseMsg;
    msg.ResText("服务器接收到你发送的消息了, 你发送的内容是: \r\n" + msg.Content +
"\r\n 点击<a href=\"http://www.baidu.com\">这里</a>, 我们有话跟你说哦");
}

```

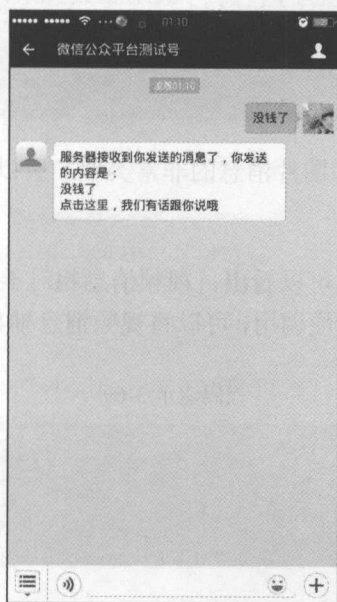


图 3-12

图片消息中的 `media_id` 是通过上传多媒体文件接口获取的, 其他的字段和文本消息类似。代码清单 3-68 是回复图片消息的方法。

代码清单 3-68

```

/// <summary>
/// 回复消息(图片)
/// </summary>
public void ResPicture(string media_id)
{

```



```
var resxml = new StringBuilder();
resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",
FromUserName);
resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",
ToUserName);
resxml.AppendFormat("<CreateTime>{0}</CreateTime>",
Utils.ConvertDateTimeInt(DateTime.Now));
resxml.Append("<MsgType><![CDATA[image]]></MsgType>");
resxml.AppendFormat(
"<Image><MediaId><![CDATA[{0}]]></MediaId></Image></xml>", media_id);
HttpContext.Current.Response.Write(resxml);
return;
}
```

语音消息的 XML 数据包和图片消息的非常类似，所以在这里就不贴出语音消息的代码了。

从视频消息的 XML 数据包可以看出，视频消息相对于图片消息、语音消息多了 Title 和 Description 两个字段。为了方便调用，可以将视频消息抽象为类，如代码清单 3-69 所示。

代码清单 3-69

```
namespace WxApi. SendEntity
{
    /// <summary>
    /// 回复视频消息实体
    /// </summary>
    public class ResVideo
    {
        /// <summary>
        /// 通过上传多媒体文件，得到的 media_id
        /// </summary>
        public string MediaId { get; set; }
        /// <summary>
        /// 视频消息的标题
        /// </summary>
        public string Title { get; set; }
        /// <summary>
        /// 视频消息的描述
        /// </summary>
        public string Description { get; set; }
    }
}
```




```
}  
}
```

代码清单 3-70 是回复视频消息的代码。

代码清单 3-70

```
/// <summary>  
/// 回复消息(视频)  
/// </summary>  
public void ResVideo(ResVideo video)  
{  
    var resxml = new StringBuilder();  
    resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",  
        FromUserName);  
    resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",  
        ToUserName);  
    resxml.AppendFormat(  
        "<CreateTime>{0}</CreateTime>",  
        Utils.ConvertDateTimeInt(DateTime.Now));  
    resxml.Append("<MsgType><![CDATA[video]]></MsgType>");  
    resxml.AppendFormat("<Video><MediaId><![CDATA[{0}]]></MediaId>",  
        video.MediaId);  
    resxml.AppendFormat("<Title><![CDATA[{0}]]></Title>", video.Title);  
    resxml.AppendFormat("<Description><![CDATA[{0}]]></Description>  
</Video></xml>", video.Description);  
    HttpContext.Current.Response.Write(resxml);  
    return;  
}
```

音乐消息实体类代码如代码清单 3-71 所示。

代码清单 3-71

```
namespace WxApi.SendEntity  
{  
    public class ResMusic  
    {  
        /// <summary>  
        /// 音乐标题  
        /// </summary>  
        public string Title { get; set; }  
        /// <summary>  
        /// 音乐描述
```




```
    /// </summary>
    public string Description { get; set; }
    /// <summary>
    /// 音乐链接
    /// </summary>
    public string MusicUrl { get; set; }
    /// <summary>
    /// 高质量音乐链接, WiFi 环境优先使用该链接播放音乐
    /// </summary>
    public string HQMusicUrl { get; set; }
    /// <summary>
    /// 缩略图的媒体 ID。通过上传多媒体文件得到的 ID
    /// </summary>
    public string ThumbMediaId { get; set; }
}
}
```

回复音乐消息的代码如代码清单 3-72 所示。

代码清单 3-72

```
/// <summary>
/// 回复消息(音乐)
/// </summary>
public void ResMusic(ResMusic music)
{
    var resxml = new StringBuilder();
    resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",
        FromUserName);
    resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",
        ToUserName);
    resxml.AppendFormat("<CreateTime>{0}</CreateTime>",
        Utils.ConvertDateTimeInt(DateTime.Now));
    resxml.Append("<MsgType><![CDATA[music]]></MsgType>");
    resxml.AppendFormat("<Music><MusicUrl><![CDATA[{0}]]></MusicUrl>",
        music.MusicUrl);
    resxml.AppendFormat("<Title><![CDATA[{0}]]></Title>", music.Title);
    resxml.AppendFormat("<HQMusicUrl><![CDATA[{0}]]></HQMusicUrl>",
        music.HQMusicUrl);
    resxml.AppendFormat("<ThumbMediaId><![CDATA[{0}]]></ThumbMediaId>",
        music.ThumbMediaId);
    resxml.AppendFormat("<Description><![CDATA[{0}]]></Description>
```



```

</Music></xml>", music.Description);
HttpContext.Current.Response.Write(resxml);
return;
}

```

最后要介绍的就是图文消息了。在回复的图文消息中支持最多 10 条的多图文。当回复的是多图文时，默认第一项为大图，其他的都是小图，如果图文数超过 10，则会发送失败。图文消息的实体类如代码清单 3-73 和代码清单 3-74 所示。

代码清单 3-73

```

namespace WxApi.SendEntity
{
    public class ResArticle
    {
        /// <summary>
        /// 消息的标题
        /// </summary>
        public string Title { get; set; }
        /// <summary>
        /// 消息的描述
        /// </summary>
        public string Description { get; set; }
        /// <summary>
        /// 图片链接，支持 JPG、PNG 格式，较好的效果为大图 360px×200px、小图
        /// 200px×200px
        /// </summary>
        public string PicUrl { get; set; }
        /// <summary>
        /// 单击图文消息跳转链接
        /// </summary>
        public string Url { get; set; }
    }
}

```

代码清单 3-74

```

/// <summary>
/// 回复图文
/// </summary>
public void ResArticles(List<ResArticle> artList)

```



```
{  
    var resxml = new StringBuilder();  
    resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",  
        FromUserName);  
    resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",  
        ToUserName);  
    resxml.AppendFormat("<CreateTime>{0}</CreateTime>",  
        Utils.ConvertDateTimeInt(DateTime.Now));  
    resxml.Append("<MsgType><![CDATA[news]]></MsgType>");  
    resxml.AppendFormat("<ArticleCount>{0}</ArticleCount><Articles>",  
        artList.Count);  
    foreach (var article in artList)  
    {  
        resxml.AppendFormat("<item><Title><![CDATA[{0}]]></Title>",  
            article.Title);  
        resxml.AppendFormat("<PicUrl><![CDATA[{0}]]></PicUrl>",  
            article.PicUrl);  
        resxml.AppendFormat("<Url><![CDATA[{0}]]></Url>", article.Url);  
        resxml.AppendFormat("<Description><![CDATA[{0}]]></Description>  
</item>", article.Description);  
    }  
    resxml.Append("</Articles></xml>");  
    HttpContext.Current.Response.Write(resxml);  
    return;  
}
```

需要注意的是，在调用的过程中，如果发送的图文消息只有一条，则标题和描述都会显示；如果大于一条，则每条图文的描述就不显示了，如图 3-13 所示。

3.4.2 客服接口

客服接口，顾名思义，用于客户服务的接口。当用户主动发消息给公众号的时候（包括发送信息、单击自定义菜单、订阅事件、扫描二维码事件、支付成功事件、用户维权），微信将会把消息以 XML 数据包的形式推送给开发者的服务器，开发者可在 48 小时内调用客服消息接口，通过 POST 一个 JSON 数据包来发送消息给普通用户，在 48 小时内不限制发送次数。此接口主要用于客户等有人工消息处理环节的功能，方便开发者为用户提供更加优质的服务。

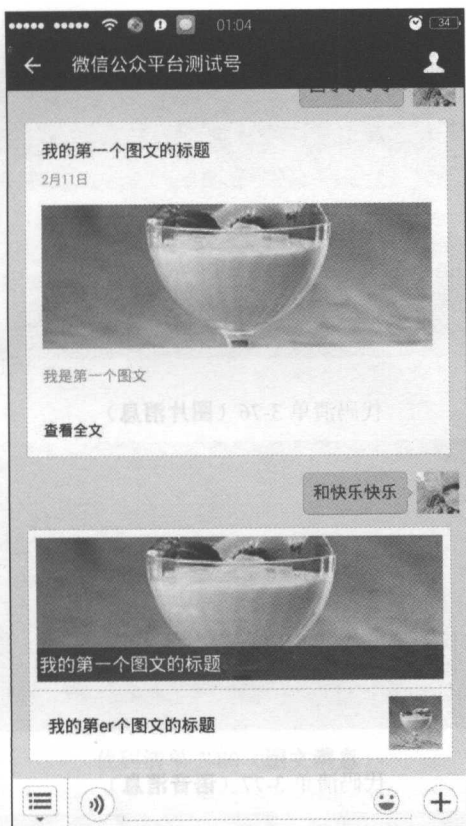


图 3-13

和发送被动响应消息类似，客服接口可发送的消息类型包括文本、语音、视频、音乐和图文，区别是客服消息接口是以 HTTP 的 POST 进行请求的，数据格式是 JSON。另外，客服消息在接收到用户的消息后，48 小时之内都是可以不限回复次数的，并且客服接口可以设置用户账户和客服头像；而被动响应消息只能在接收到用户请求的 5 秒之内进行回复，且只能回复一次。

客服接口发消息的接口地址如下：

https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

各消息类型所需的 JSON 数据包如代码清单 3-75 到代码清单 3-80 所示。



代码清单 3-75 (文本消息)

```
{
  "touser": "openid",
  "msgtype": "text",
  "text": {
    "content": "Hello World"
  }
}
```

代码清单 3-76 (图片消息)

```
{
  "touser": "openid",
  "msgtype": "image",
  "image": {
    "media_id": "MEDIA_ID"
  }
}
```

代码清单 3-77 (语音消息)

```
{
  "touser": "openid",
  "msgtype": "voice",
  "voice": {
    "media_id": "MEDIA_ID"
  }
}
```

代码清单 3-78 (视频消息)

```
{
  "touser": "openid",
  "msgtype": "video",
  "video": {
    "media_id": "MEDIA_ID",
  }
}
```




```
"thumb_media_id":"MEDIA_ID",  
"title":"TITLE",  
"description":"DESCRIPTION"  
}  
}
```

代码清单 3-79 (音乐消息)

```
{  
  "touser":"openid",  
  "msgtype":"music",  
  "music":  
  {  
    "title":"MUSIC_TITLE",  
    "description":"MUSIC_DESCRIPTION",  
    "musicurl":"MUSIC_URL",  
    "hgmusicurl":"HQ_MUSIC_URL",  
    "thumb_media_id":"THUMB_MEDIA_ID"  
  }  
}
```

代码清单 3-80 (图文消息)

```
{  
  "touser":"openid",  
  "msgtype":"news",  
  "news":{  
    "articles": [  
      {  
        "title":"Happy Day",  
        "description":"Is Really A Happy Day",  
        "url":"URL",  
        "picurl":"PIC_URL"  
      },  
      {  
        "title":"Happy Day",  
        "description":"Is Really A Happy Day",  
        "url":"URL",  
        "picurl":"PIC_URL"  
      }  
    ]  
  }  
}
```



```
    ]  
  }  
}
```

从数据格式中可以看出，在文本、图片、语音的消息体中，表示消息内容的只有一个字段，如文本消息的 `content`，图片和语音的 `media_id`。而视频、音乐和图文消息的内容包括多个字段，为了调用方便，需要将多个字段抽象成类。下面是各消息类型对应的实体类（见代码清单 3-81 至代码清单 3-83）。

代码清单 3-81（视频消息）

```
public class CustomVideo  
{  
    /// <summary>  
    /// 媒体 ID  
    /// </summary>  
    public string media_id { get; set; }  
    /// <summary>  
    /// 缩略图 ID  
    /// </summary>  
    public string thumb_media_id { get; set; }  
    /// <summary>  
    /// 标题  
    /// </summary>  
    public string title { get; set; }  
    /// <summary>  
    /// 描述  
    /// </summary>  
    public string description { get; set; }  
}
```

代码清单 3-82（音乐消息实体）

```
public class CustomMusic  
{  
    /// <summary>  
    /// 音乐链接  
    /// </summary>  
    public string musicurl { get; set; }  
}
```



```
/// <summary>
/// 高品质音乐链接, WiFi 环境优先使用该链接播放音乐
/// </summary>
public string hqmusicurl { get; set; }
/// <summary>
/// 缩略图 ID
/// </summary>
public string thumb_media_id { get; set; }
/// <summary>
/// 标题
/// </summary>
public string title { get; set; }
/// <summary>
/// 描述
/// </summary>
public string description { get; set; }
}
```

代码清单 3-83 (图文消息实体)

```
public class CustomArticles
{
    public List<CustomArticle> articles { get; set; }
}

public class CustomArticle
{
    /// <summary>
    /// 标题
    /// </summary>
    public string title { get; set; }
    /// <summary>
    /// 描述
    /// </summary>
    public string description { get; set; }
    /// <summary>
    /// 图文链接
    /// </summary>
    public string url { get; set; }
}
```



```
/// <summary>
/// 图片链接
/// </summary>
public string picurl { get; set; }
}
```

基础工作准备完毕，就可以编写调用客服接口的代码了，如代码清单 3-84 所示。

代码清单 3-84

```
/// <summary>
/// 客服消息接口
/// </summary>
public class CustomerServices
{
    /// <summary>
    /// http 请求方式: POST
    /// </summary>
    private static string url =
        "https://api.weixin.qq.com/cgi-bin/message/custom/send
        ?access_token={0}";
    /// <summary>
    /// 发送文本
    /// </summary>
    public static ErrorEntity SendText(string openid, string content, string
        accessToen)
    {
        var json = new
        {
            touser = openid,
            msgtype = "text",
            text = new
            {
                content = content
            }
        };
        return Send(json, accessToen);
    }
    /// <summary>
```




```
/// 发送图片
/// </summary>
public static ErrorEntity SendImg(string openid, string media_id, string
accessToen)
{
    var json = new
    {
        touser = openid,
        msgtype = "image",
        image = new
        {
            media_id = media_id
        }
    };
    return Send(json, accessToen);
}
/// <summary>
/// 发送语音消息
/// </summary>
public static ErrorEntity SendVoice(string openid, string media_id,
string accessToen)
{
    var json = new
    {
        touser = openid,
        msgtype = "voice",
        voice = new
        {
            media_id = media_id
        }
    };
    return Send(json, accessToen);
}
/// <summary>
/// 发送视频消息
/// </summary>
public static ErrorEntity SendVideo(string openid, CustomVideo Video,
string accessToen)
```




```
{
    var json = new
    {
        touser = openid,
        msgtype = "video",
        video = Video
    };
    return Send(json, accessToen);
}
/// <summary>
/// 发送音乐消息
/// </summary>
public static ErrorEntity SendMusic(string openid, CustomMusic music,
string accessToen)
{
    var json = new
    {
        touser = openid,
        msgtype = "music",
        music = music
    };
    return Send(json, accessToen);
}
/// <summary>
/// 发送图文消息
/// </summary>
public static ErrorEntity SendArticle(string openid, CustomArticle
article, string accessToen)
{
    var json = new
    {
        touser = openid,
        msgtype = "news",
        news = article
    };
    return Send(json, accessToen);
}
```



```
private static ErrorEntity Send(object obj, string accessToen)
{
    return Utils.PostResult<ErrorEntity>(obj, string.Format(url,
accessToen));
}
}
```

3.5 高级群发接口

从微信公众平台喊出那句“再小的个体也需要品牌”的时候，消息群发就注定成为微信营销人员的刚需功能。在公众平台的后台，为订阅号提供了每天一条的群发权限，为服务号提供了每月4条的群发权限。而对于具备开发能力的公众号运营者，则可以通过高级群发接口，实现更灵活的群发能力。

需要注意以下几点：

- 对于**认证订阅号**，群发接口**每天可成功调用一次**，此次群发可选择发送给全部用户或某个分组。
- 对于**认证服务号**，虽然开发者使用高级群发接口的**每日调用限制为100次**，但是用户**每月只能接收4条**。所以，无论在公众平台网站上，还是使用接口群发，用户每月只能接收4条群发消息，多于4条的群发将对该用户发送失败。
- 具备微信支付权限的公众号，在使用群发接口上传、群发图文消息类型时，可使用[<a>](#)标签加入外链。
- 开发者可以使用预览接口校对消息样式和排版，通过预览接口可发送编辑好的消息给指定用户校验效果。

3.5.1 上传图文消息素材

调用群发接口发送图文消息时，需要先将图文素材上传到微信服务器，上传成功后，会返回一个 `media_id`，然后再调用群发图文的接口。这一点类似于客服接口发送媒体消息（这里上传的图文素材和素材管理中的图文素材并不是通用的）。



上传图文消息素材的接口地址如下：

https://api.weixin.qq.com/cgi-bin/media/uploadnews?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

在请求的时候，此接口需要 POST 的 JSON 数据与素材管理接口的新增永久图文接口是一致的，在此就不赘述了。调用接口时的返回实体如代码清单 3-85 所示。

代码清单 3-85

```
namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// 群发时，上传媒体消息时的返回实体
    /// </summary>
    public class GroupUploadEntity:ErrorEntity
    {
        /// <summary>
        /// 媒体文件类型，分别有图片（image）、语音（voice）、视频（video）和缩略图
        /// （thumb）
        /// </summary>
        public string type { get; set; }
        /// <summary>
        /// 媒体文件/图文消息上传后获取的唯一标识
        /// </summary>
        public string media_id { get; set; }
    }
}
```

实体创建完成后，在 WxApi 项目中新建类 GroupSend，用于封装群发接口相关的操作。在类中创建方法 UploadNew，该方法的功能是上传图文消息，并获取请求成功后的返回信息，如代码清单 3-86 所示。

代码清单 3-86

```
public class GroupSend
{
    /// <summary>
    /// 上传图文消息
    /// </summary>
```



```
public static GroupUploadEntity UploadNew(List<Article> artList, string
accessToken)
{
    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/media/uploadnews
?access_token={0}", accessToken);
    var json = new
    {
        articles = artList
    };
    return Utils.PostResult<GroupUploadEntity>(json, url);
}
```

3.5.2 根据分组进行群发

公众号为认证的服务号提供了按用户分组和 openid 列表两种群发方式，认证的订阅号只支持按用户分组进行群发，这一点是开发者需要特别注意的。

根据分组群发的接口地址如下：

https://api.weixin.qq.com/cgi-bin/message/mass/sendall?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例如代码清单 3-87 到代码清单 3-91 所示。

代码清单 3-87（图文消息）

```
{
    "filter":{
        "is_to_all":false,
        "group_id":"2"
    },
    "mpnews":{
        "media_id":"123dsdajkasd231jhksad"
    },
    "msgtype":"mpnews"
}
```




代码清单 3-88 (文本消息)

```
{
  "filter":{
    "is_to_all":false,
    "group_id":"2"
  },
  "text":{
    "content":"CONTENT"
  },
  "msgtype":"text"
}
```

代码清单 3-89 (语音消息)

```
{
  "filter":{
    "is_to_all":false,
    "group_id":"2"
  },
  "voice":{
    "media_id":"123dsdajkasd231jhksad"
  },
  "msgtype":"voice"
}
```

代码清单 3-90 (图片消息)

```
{
  "filter":{
    "is_to_all":false
    "group_id":"2"
  },
  "image":{
    "media_id":"123dsdajkasd231jhksad"
  },
  "msgtype":"image"
}
```

代码清单 3-91 (视频消息)

```
{
  "filter":{
```




```
"is_to_all":false,
"group_id":"2"
},
"mpvideo":{
  "media_id":"IhdaAQXuvJtGzwWC0abfXnzeezf00NgPK6AQYShD8RQYMTtfzbLd
Biv2XJc",
},
"msgtype":"mpvideo"
}
```

请注意，群发接口发送视频消息的数据包和客服接口的数据包是有很大区别的。从代码清单 3-90 中可以看出，mpvideo 的值是一个 media_id 对象，而 media_id 对象的值是需要调用接口进行上传后获取到的。调用接口成功后，返回的实体是代码清单 3-84 所示的类 GroupUploadEntity，接口地址如下：

https://file.api.weixin.qq.com/cgi-bin/media/uploadvideo?access_token=ACCESS_TOKEN

HTTP 请求方式：POST

请求示例如代码清单 3-92 所示。

代码清单 3-92

```
{
  "media_id": "rF4UdIMfYK3efUfyoddYRMU50zMiRmmt_lupYh_SzrcW5Gaheq05p_
lHuOTQ",
  "title": "TITLE",
  "description": "Description"
}
```

上述代码清单中的 media_id 是需要调用素材管理中的新增临时素材接口（不适用永久视频素材），将视频文件上传后，获取到的 media_id 获取到的。

在类 GroupSend 中，添加方法 UploadVideo，其功能是上传视频消息，并获取返回实体，如代码清单 3-93 所示。

代码清单 3-93

```
public static GroupUploadEntity UploadVideo(string media_id, string title,
string description, string accessToken)
```



```
{
    var url=
        string.Format("https://file.api.weixin.qq.com/cgi-bin/media/
uploadvideo?access_token={0}", accessToken);
    var json = new
    {
        media_id = media_id,
        title = title,
        description = description
    };
    return Utils.PostResult<GroupUploadEntity>(json, url);
}
```

在各个消息类型的 JSON 数据包中，filter 用于设定消息的接收者。is_to_all 用于设定是否向全部用户发送，值为 true 或 false，选择 true，则该消息群发给所有用户；选择 false，则可根据 group_id 发送给指定群组的用户。is_to_all 为 true 且成功群发时，会使得此次群发进入历史消息列表。为防止异常，认证订阅号在一天内，只能使用 is_to_all 为 true 进行一次群发，或者在公众平台官网群发（不管本次群发是对全体还是对某个分组）一次。以避免一天内有两条群发进入历史消息列表。类似地，服务号在一个月内，使用 is_to_all 为 true 群发的次数，加上公众平台官网群发的次数，最多只能是 4 次。**设置 is_to_all 为 false 时可以多次群发，但每个用户只会收到最多 4 条，且这些群发不会进入历史消息列表。**group_id 表示的是群发到的分组的 group_id，参考用户管理中的用户分组接口，若 is_to_all 值为 true，则可不填写 group_id。msgtype 表示的是消息类型，文本消息数据包中的 content 表示的是要发送的文本内容。

由于发送各个消息类型的接口地址和调用方式都是一样的，不一样的只是数据包的格式与内容，因此我们只需在不同的方法中组织好 JSON 数据包，然后调用统一的发送方法即可。在类 GroupSend 中添加私有方法 BaseSend，如代码清单 3-94 所示。

代码清单 3-94

```
/// <summary>
/// 基础发送接口
/// </summary>
/// <param name="obj">json 对象</param>
/// <param name="accessToken">accessToken</param>
/// <param name="sendtype">群发类型，1 为按分组群发，2 为按 openid 列表群发，3 为
```



```
/// 预览接口。默认为按分组群发</param>
private static GroupSendEntity BaseSend(object obj, string accessToken, int
sendtype= 1)
{
    string url = null;
    switch (sendtype)
    {
        //分组群发
        case 1:url =
            string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/
sendall?access_token={0}",accessToken);break;
        //openid列表群发
        case 2: url =
            string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/send?
access_token={0}", accessToken); break;
        //预览接口
        case 3: url =
            string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/
preview?access_token={0}",accessToken); break;
        default: url =
            string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/
sendall?access_token={0}",accessToken); break;
    }
    return Utils.PostResult<GroupSendEntity>(obj, url);
}
```

在 BaseSend 方法中, 根据传入的 sendtype 的值, 调用不同的接口。此方法封装了“按分组 ID 群发”、“按 openid 列表群发”, 以及为了让开发者查看消息的样式和模板的预览接口。

BaseSend 方法的返回类型是 GroupSendEntity, 定义如代码清单 3-95 所示。

代码清单 3-95

```
namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// 群发接口返回值
    /// </summary>
```



```
public class GroupSendEntity : ErrorEntity
{
    /// <summary>
    /// 消息 ID
    /// </summary>
    public string msg_id { get; set; }
}
```

请注意：在返回成功时，意味着群发任务提交成功，但并不意味着此时群发已经结束。所以，仍有可能在后续的发送过程中出现异常情况导致用户未收到消息，如消息有时会进行审核、服务器不稳定等。此外，群发任务一般需要较长的时间才能全部发送完毕，请耐心等待。

定义好基本发送方法后，开发者只需根据不同的消息类型，组织不同的 JSON 对象，然后调用 BaseSend 即可（见代码清单 3-96 至代码清单 3-100）。

代码清单 3-96（图文消息）

```
/// <summary>
/// 按分组群发图文消息，isall 为 true 时，说明群发所有用户，此时 group_id 可为空；
/// 否则，根据 group_id 进行群发
/// </summary>
/// <param name="media_id">图片的媒体 ID</param>
/// <param name="accessToken">accessToken</param>
/// <param name="isall">是否群发所有用户</param>
/// <param name="group_id">分组 ID</param>
public static GroupSendEntity SendArticleByGroup(string media_id, string
accessToken, bool isall = true, string group_id = "")
{
    var json = new
    {
        filter = new
        {
            is_to_all = isall,
            group_id = group_id
        },
        mpnews = new
    {

```




```
        media_id = media_id
    },
    msgtype = "mpnews"
};
return BaseSend(json, accessToken);
}
```

代码清单 3-97 (文本消息)

```
public static GroupSendEntity SendTextByGroup(string content, string
accessToken, bool isall = true, string group_id="")
{
    var json = new
    {
        filter = new
        {
            is_to_all = isall,
            group_id = group_id
        },
        text = new
        {
            content = content
        },
        msgtype = "text"
    };
    return BaseSend(json, accessToken);
}
```

代码清单 3-98 (语音消息)

```
public static GroupSendEntity SendVoiceByGroup(string media_id, string
accessToken, bool isall = true, string group_id = "")
{
    var json = new
    {
        filter = new
        {
            is_to_all = isall,
            group_id = group_id
        },
        voice = new
```




```
{
    media_id = media_id
},
msgtype = "voice"
};
return BaseSend(json, accessToken);
}
```

代码清单 3-99 (图片消息)

```
public static GroupSendEntity SendImgByGroup(string media_id, string
accessToken, bool isall = true, string group_id = "")
{
    var json = new
    {
        filter = new
        {
            is_to_all = isall,
            group_id = group_id
        },
        image = new
        {
            media_id = media_id
        },
        msgtype = "image"
    };
    return BaseSend(json, accessToken, 1);
}
```

代码清单 3-100 (视频消息)

```
public static GroupSendEntity SendVideoByGroup(string media_id, string
accessToken, bool isall = true, string group_id = "")
{
    var json = new
    {
        filter = new
        {
            is_to_all = isall,
            group_id = group_id
        },
        video = new
        {
            media_id = media_id
        },
        msgtype = "video"
    };
    return BaseSend(json, accessToken, 1);
}
```



```
    },
    mpvideo = new
    {
        media_id = media_id
    },
    msgtype = "mpvideo"
};
return BaseSend(json, accessToken);
}
```

3.5.3 根据 openid 列表群发和预览消息

根据 openid 列表群发接口需要传入的参数是 openid 数组，POST 的数据格式与根据分组群发相似，只需将分组群发中的 filter 对象换成 touser 数组即可。表 3-4 所示的是 3 种接口的 JSON 数据包的格式。

表 3-4

按分组群发文本消息	按 openid 列表群发文本消息	预览文本消息接口
<pre>{ "filter":{ "is_to_all":false, "group_id":"2" }, "msgtype":"text", "text":{"content":"CONTENT"} }</pre>	<pre>{ "touser":["openid", "openid"], "msgtype": "text", "text":{"content":"CONTENT"} }</pre>	<pre>{ "touser":"openid", "msgtype":"text", "text":{"content":"CONTENT"} }</pre>

由于按 openid 列表群发和按分组群发的 JSON 数据包类似，限于篇幅，在此不再赘述。而预览消息接口和按 openid 列表群发接口的数据包的唯一区别是 touser 对象。预览接口的 touser 的类型是字符串，表示的是单个用户的 openid，即要接收消息的用户。按 openid 群发接口的 touser 的类型是字符串数组，表示用户列表。既然区别不大，那么在程序设计的时候，开发者可以将 touser 定义为 object 类型。在调用的时候，如果是调用预览接口，则传入字符串类型；否则传入字符串数组。在处理的方法中判断传入的 touser 是否是数组，如果是，则说明是调用的群发接口；否则是预览接口。代码清单 3-101 是具体的实现代码。

代码清单 3-101

```
#region 按 openid 列表群发和预览接口【订阅号不可用，服务号认证后可用】
```



```
/// <summary>
/// 按用户列表群发文本消息
/// </summary>
/// <param name="content">群发内容</param>
/// <param name="accessToken">accessToken</param>
/// <param name="touser">如果为数组，则表示 openid 列表，调用的是群发接口；否则表
/// 示 openid，调用的是预览接口</param>
public static GroupSendEntity SendTextByOpenID(string content, string
accessToken, object touser)
{
    var json = new
    {
        touser = touser,
        text = new
        {
            content = content
        },
        msgtype = "text"
    };
    return BaseSend(json, accessToken, touser.GetType().IsArray?2:3);
}
/// <summary>
/// 按用户列表群发图片消息
/// </summary>
/// <param name="media_id">图片媒体 ID</param>
/// <param name="accessToken">accessToken</param>
/// <param name="touser">如果为数组，则表示 openid 列表，调用的是群发接口；否则表示
/// openid，调用的是预览接口</param>
public static GroupSendEntity SendImgByOpenID(string media_id, string
accessToken, object touser)
{
    var json = new
    {
        touser = touser,
        image = new
        {
            media_id = media_id
        },
    },
```



```
        msgtype = "image"
    };
    return BaseSend(json, accessToken, touser.GetType().IsArray ? 2 : 3);
}
/// <summary>
/// 按用户列表群发语音消息
/// </summary>
/// <param name="media_id">语音媒体 ID</param>
/// <param name="accessToken">accessToken</param>
/// <param name="touser">如果为数组，则表示 openid 列表，调用的是群发接口；否则表
/// 示 openid，调用的是预览接口</param>
public static GroupSendEntity SendVoiceByOpenID(string media_id, string
accessToken, object touser)
{
    var json = new
    {
        touser = touser,
        voice = new
        {
            media_id = media_id
        },
        msgtype = "voice"
    };
    return BaseSend(json, accessToken, touser.GetType().IsArray ? 2 : 3);
}
/// <summary>
/// 按用户列表群发图文消息
/// </summary>
/// <param name="media_id">图文 ID</param>
/// <param name="accessToken">accessToken</param>
/// <param name="touser">如果为数组，则表示 openid 列表，调用的是群发接口；否则表
/// 示 openid，调用的是预览接口</param>
public static GroupSendEntity SendArticleByOpenID(string media_id, string
accessToken, object touser)
{
    var json = new
    {
        touser = touser,
```




```
        mpnews = new
        {
            media_id = media_id
        },
        msgtype = "mpnews"
    };
    return BaseSend(json, accessToken, touser.GetType().IsArray ? 2 : 3);
}
/// <summary>
/// 按用户列表群发视频消息
/// </summary>
/// <param name="media_id">视频 ID</param>
/// <param name="accessToken">accessToken</param>
/// <param name="touser">如果为数组，则表示 openid 列表，调用的是群发接口；否则表
/// 示 openid，调用的是预览接口</param>
public static GroupSendEntity SendVideoByOpenID(string media_id, string
accessToken, object touser)
{
    var json = new
    {
        touser = touser,
        mpvideo = new
        {
            media_id = media_id
        },
        msgtype = "mpvideo"
    };
    return BaseSend(json, accessToken, touser.GetType().IsArray ? 2 : 3);
}
#endregion
```

3.5.4 事件推送群发结果

由于群发任务提交后，该群发任务可能在一定时间后才完成，因此，群发接口调用时，仅会给出群发任务是否提交成功的提示。若群发任务提交成功，则在群发任务结束时，会向开发者在公众平台填写的开发者 URL（callback URL）推送事件。这里推送事件可以理解为一个特殊的用户发送了一条特殊的消息，而开发者只需按照 3.3 节（接收消息）的方法处理这条消息即可。



推送的 XML 接口如代码清单 3-102 所示。

代码清单 3-102

```
<xml>
  <ToUserName><![CDATA[gh_3e8adccde292]]></ToUserName>
  <FromUserName><![CDATA[or5Gjj1_eiZoUpGozMo7dbBJ362A]]></FromUserName>
  <CreateTime>1394524295</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[MASSENDJOBFINISH]]></Event>
  <MsgID>1988</MsgID>
  <Status><![CDATA[sendsuccess]]></Status>
  <TotalCount>100</TotalCount>
  <FilterCount>80</FilterCount>
  <SentCount>75</SentCount>
  <ErrorCount>5</ErrorCount>
</xml>
```

从消息的结构上可以看出,此消息是事件消息,事件类型是“MASSENDJOBFINISH”,其中 MsgID 是群发消息的 ID, Status 表示的是群发的状态,为“send success”、“send fail”或“err(num)”。但在 send success 时,也有可能因用户拒收公众号的消息、系统错误等原因造成少量用户接收失败。err(num)是审核失败的具体原因,可能的情况如下:

- err(10001)涉嫌广告。
- err(20001)涉嫌政治。
- err(20004)涉嫌社会。
- err(20002)涉嫌色情。
- err(20006)涉嫌违法犯罪。
- err(20008)涉嫌欺诈。
- err(20013)涉嫌版权。
- err(22000)涉嫌互推(互相宣传)。
- err(21000)涉嫌其他。



TotalCount 表示此次发送任务发送的粉丝数。如果是按分组群发的,则表示 group_id 下的粉丝数;如果按 openid 列表,则表示是列表中的数量。FilterCount 表示过滤后,准备发送的粉丝数,原则上,FilterCount = SentCount + ErrorCount。SentCount 表示发送成功的粉丝数,ErrorCount 表示发送失败的粉丝数。由于群发接口并未设置过滤条件的接口,因此,FilterCount 目前只能过滤用户接收已超 4 条的用户。

根据 XML 结构新建类 GroupJobEventMsg,如代码清单 3-103 所示。

代码清单 3-103

```
public class GroupJobEventMsg:EventArgs
{
    /// <summary>
    /// 群发的消息 ID
    /// </summary>
    public string MsgID { get; set; }
    /// <summary>
    /// 发送状态
    /// </summary>
    public string Status { get; set; }
    /// <summary>
    /// group_id 下粉丝数; 或者 openid_list 中的粉丝数
    /// </summary>
    public string TotalCount { get; set; }
    /// <summary>
    /// 过滤 (过滤是指特定地区、性别的过滤, 用户设置拒收的过滤, 用户接收已超 4 条的过滤)
    /// 后, 准备发送的粉丝数。原则上, FilterCount = SentCount + ErrorCount
    /// </summary>
    public string FilterCount { get; set; }
    /// <summary>
    /// 发送成功的粉丝数
    /// </summary>
    public string SentCount { get; set; }
    /// <summary>
    /// 发送失败的粉丝数
    /// </summary>
    public string ErrorCount { get; set; }
}
```



然后再在枚举 EventType 中添加项 MASSENDJOBFINISH。在类 MsgFactory 的 LoadMsg 方法中，在图 3-14 所示的位置添加如代码清单 3-104 所示的代码。

代码清单 3-104

```
case EventType.MASSENDJOBFINISH:
msg = Utils.ConvertObj<GroupJobEventMsg>(postStr);break;
```

```
case MsgType.EVENT://事件类型
{
    eventtype = (EventType)Enum.Parse(typeof(EventType), xdoc.Element(
switch (eventtype)
{
    //订阅与取消订阅事件
    case EventType.UNSUBSCRIBE: msg = Utils.ConvertObj<SubEventMsg>(
    //上报地理位置事件
    case EventType.LOCATION: msg = Utils.ConvertObj<LocationEventM
    case EventType.MASSENDJOBFINISH:
        msg = Utils.ConvertObj<GroupJobEventMsg>(postStr);break;
```

图 3-14

最后，需要在项目 WxTest 的 wx.ashx 处理程序中添加处理群发消息的代码，如代码清单 3-105 所示。

代码清单 3-105

```
private void GroupJobHandler(BaseMsg baseMsg)
{
    //此处编写业务处理代码
}
```

另外，需要将此处理方法在程序加载运行时与处理消息的业务逻辑进行绑定，如代码清单 3-106 所示。

代码清单 3-106

```
#region 群发消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new MsgHandlerEntity
{
    MsgType = MsgType.EVENT,
    EventType = EventType.MASSENDJOBFINISH,
```



```
        Action = GroupJobHandler  
    });  
#endregion
```

3.5.5 查询群发消息发送状态

前面提到过，在调用群发接口成功后，只是代表这个群发任务提交成功了，并不能从接口的返回信息判断是否发送成功。但开发者可以通过接口查询群发消息的发送状态。接口地址如下：

https://api.weixin.qq.com/cgi-bin/message/mass/get?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

数据示例：{"msg_id": "201053012"}。

msg_id 是群发消息成功后返回的。开发者只需每次群发时，将返回的 msg_id 保存下来，当需要查询群发消息发送状态时，直接将 msg_id 以 JSON 格式 POST 接口即可。具体实现如代码清单 3-107 所示。

代码清单 3-107

```
public static GroupStatus QueryStatus(string msg_id, string accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/get?access_token={0}", accessToken);  
    var json = new { msg_id = msg_id };  
    return Utils.PostResult<GroupStatus>(json, url);  
}
```

方法 QueryStatus 返回值类型是 GroupStatus，是请求接口返回的数据的实体映射，如代码清单 3-108 所示。

代码清单 3-108

```
public class GroupStatus : ErrorEntity  
{  
    /// <summary>
```




```
/// 群发消息后返回的消息 ID
/// </summary>
public string msg_id { get; set; }
/// <summary>
/// 消息发送后的状态, SEND_SUCCESS 表示发送成功
/// </summary>
public string msg_status { get; set; }
}
```

3.5.6 删除群发

为了解决由于误操作群发消息而给用户带来不好的用户体验, 公众平台提供了删除群发消息的接口。请注意, 只有已经发送成功的消息才能删除。删除消息只是将消息的图文详情页失效; 已经收到的用户, 还是能在其本地看到消息卡片的。另外, 删除群发消息只能删除图文消息和视频消息; 其他类型的消息一经发送, 无法删除。

删除群发的接口地址如下:

https://api.weixin.qq.com/cgi-bin/message/mass/delete?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

数据示例: `{"msg_id":30124}`。

具体实现如代码清单 3-109 所示。

代码清单 3-109

```
/// <summary>
/// 删除群发
/// </summary>
/// <param name="msg_id">群发消息 ID</param>
public static ErrorEntity Delete(string msg_id, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/message/mass/delete?access_token={0}", accessToken);
    var json = new {msg_id = msg_id};
}
```



```
return Utils.PostResult<ErrorEntity>(json, url);
```

删除成功后，当用户查看消息的详细信息时，将显示“该内容已被发布者删除”。

3.6 业务通知模板消息

模板消息仅用于公众号向用户发送重要的服务通知，只能用于符合其要求的服务场景中，如信用卡刷卡通知、商品购买成功通知等。不支持广告等营销类消息以及其他所有可能对用户造成骚扰的消息。

关于使用规则，请注意：

- 在选择模板之前，需要选择公众账号服务所处的两个行业，每月可更改一次所选行业。
- 在所选择行业的模板库中选用已有的模板进行调用。
- 每个账号可以同时使用 15 个模板。

3.6.1 设置公众号所属行业

公众平台后台提供了设置行业的入口，每月可修改一次，账号仅可使用所属行业相关的模板。为方便第三方开发者，提供通过接口调用的方式来修改账号所属行业。接口地址如下：

https://api.weixin.qq.com/cgi-bin/template/api_set_industry?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

每一个公众号可以设置一个主营行业和一个副营行业，所以在调用接口时，需要 POST 两个行业的 ID，POST 数据实例如下：

```
{"industry_id1": "1", "industry_id2": "4" }
```

其中 industry_id1 表示的是主营行业编号，industry_id2 是副营行业编号。



行业代码可从代码清单 3-110 所示的 JSON 字符串中获得。

代码清单 3-110

```
{
  "IT 科技": {
    "互联网/电子商务": 1,
    "IT 软件与服务": 2,
    "IT 硬件与设备": 3,
    "电子技术": 4,
    "通信与运营商": 5,
    "网络游戏": 6
  },
  "金融业": {
    "银行": 7,
    "基金|理财|信托": 8,
    "保险": 9
  },
  "餐饮": {
    "餐饮": 10
  },
  "酒店旅游": {
    "酒店": 11,
    "旅游": 12
  },
  "运输与仓储": {
    "快递": 13,
    "物流": 14,
    "仓储": 15
  },
  "教育": {
    "培训": 16,
    "院校": 17
  },
  "政府与公共事业": {
    "学术科研": 18,
    "交警": 19,
    "博物馆": 20,
    "公共事业|非盈利机构": 21
  }
}
```



```
},  
"医药护理": {  
    "医药医疗": 22,  
    "护理美容": 23,  
    "保健与卫生": 24  
},  
"交通工具": {  
    "汽车相关": 25,  
    "摩托车相关": 26,  
    "火车相关": 27,  
    "飞机相关": 28  
},  
"房地产": {  
    "建筑": 29,  
    "物业": 30  
},  
"消费品": {  
    "消费品": 31  
},  
"商业服务": {  
    "法律": 32,  
    "会展": 33,  
    "中介服务": 34,  
    "认证": 35,  
    "审计": 36  
},  
"文体娱乐": {  
    "传媒": 37,  
    "体育": 38,  
    "娱乐休闲": 39  
},  
"印刷": {  
    "印刷": 40  
},  
"其它": {  
    "其它": 41  
}  
}
```




为了方便调用,这里参考 3.1.1 节全局返回码的处理方式。将上述代码保存在文本文件 `IndustryCode` 中,然后把文本文件添加到资源文件 `Code` 中。

新建类 `TemplateNotice`,在类中添加一个静态的私有字段 `_IndustryList` 和静态的公共属性 `IndustryList`。在 `IndustryList` 的 `get` 访问器中,首先判断 `_IndustryList` 的值是否为空。如果为空,则从资源文件中获取 JSON 字符串,并转换成对象,赋值给 `_IndustryList`,最后直接返回 `_IndustryList`,如代码清单 3-111 所示。

代码清单 3-111

```
private static Dictionary<string, Dictionary<string, int>> _IndustryList;
public static Dictionary<string, Dictionary<string, int>> IndustryList
{
    get
    {
        if (_IndustryList == null)
            _IndustryList =
                JsonConvert.DeserializeObject<Dictionary<string, Dictionary
<string, int>>>(Code.IndustryCode);
        return _IndustryList;
    }
}
```

获取到了行业 ID 后,就可以设置公众号所属行业了。如图 3-15 所示,主营行业的两个下拉选择框与副营行业的两个下拉列表框都是从 `TemplateNotice` 类的 `IndustryList` 属性中获取到的。当选择完成后,单击“确认”按钮后,调用设置所属行业的接口,即可设置成功。

图 3-15



设置公众号所属行业的代码如代码清单 3-112 所示。

代码清单 3-112

```
public static ErrorEntity SetIndustry(string id1, string id2, string
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/template/
api_set_industry?access_token={0}", accessToken);
    var json = new {industry_id1 = id1, industry_id2 = id2 };
    return Utils.PostResult<ErrorEntity>(json, url);
}
```

3.6.2 获取模板 ID

在发送模板消息之前，需要先获取模板的 ID。在公众平台的模板库中有很多模板，每个模板都有一个唯一的模板编号，格式如“OPENTM*****”和“TM*****”。公众号在使用模板之前，必须从模板库中选择相应的模板添加到“我的模板”中。那么问题来了：添加到“我的模板”中的模板会生成一个模板 ID，这个 ID 是相对于公众号唯一的。也就是说多个公众号选择相同模板编号的模板，生成的 ID 是不一样的，甚至当我们不小心将“我的模板”中的某个模板删除后，再次添加时生成的 ID 和删除前的也是不一样的，这给我们的程序带来了很大的不便。庆幸的是，官方提供了可以根据模板编号获取模板 ID 的接口。在我们设计程序时，只需保存所使用的模板编号即可，在需要发送模板消息时，可调用接口将模板编号转换成模板 ID。接口地址如下：

https://api.weixin.qq.com/cgi-bin/template/api_add_template?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例：{"template_id_short":"TM00015"}。

具体实现如代码清单 3-113 所示。

代码清单 3-113

```
/// <summary>
/// 获取模板 ID
/// </summary>
```



```
/// <param name="templateNo">模板编号</param>
/// <param name="accessToken">accessToken</param>
public static TemplateID GetTemplateID(string templateNo, string
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/template/
api_add_template?access_token={0}", accessToken);
    var json = new { template_id_short = templateNo };
    return Utils.PostResult<TemplateID>(json, url);
}
```

方法中返回类型 `TemplateID` 的定义如代码清单 3-114 所示。

代码清单 3-114

```
public class TemplateID:ErrorEntity
{
    /// <summary>
    /// 模板 ID
    /// </summary>
    public string template_id { get; set; }
}
```

模板消息接口自从开放之后，模板库也在不断地完善，但有些冷门的行业或者冷门的业务需求可能会出现在模板库中找不到适合的模板这种情况。此时，在满足一定要求的情况下，你可以为自己所在的行业贡献新模板，帮助充实模板库。在公众平台的后台“功能”→“模板消息”→“模板库”中，找到如图 3-16 所示的位置。

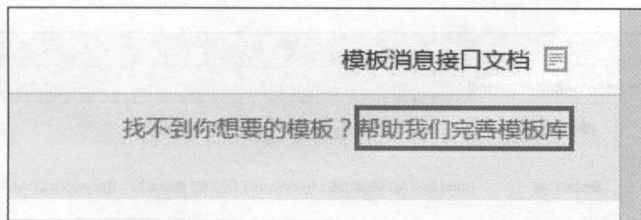


图 3-16

在添加模板消息时，请仔细阅读《模板消息申请添加前必读指引》。请勿违反公众平台



运营规则；否则可能被停用模板消息接口甚至封号。贡献新模板需要等待 7~15 天审核期，且内容可能被审核人员修改。每个公众号每月可申请新建 3 个新模板。审核通过后，模板将放入模板库以供他人使用。如图 3-17 所示的是添加模板的表单。在接口调用的时候，只需将类似 `{{first.DATA}}` 的模板参数替换掉即可。

行业

☒ IT科技/互联网|电子商务 ☐ 餐饮/餐饮

标题

请填写标题

标题需在4-12字之间，清晰详细的标题有利于通过审核

建议模板标题以“通知”或“提醒”作为结尾

内容

`{{first.DATA}}`

填写关键字 `: {{keyword1.DATA}}`

请填写关键字

填写关键字 `: {{keyword2.DATA}}`

请填写关键字

增加关键词

`{{remark.DATA}}`

内容中，类似`{{first.DATA}}`的，即为模板参数，参数的具体使用请见接口文档

标题

刷卡提醒

内容

`{{first.DATA}}`
银行卡号：`{{keyword1.DATA}}`
消费金额：`{{keyword2.DATA}}`
时间：`{{keyword4.DATA}}`
`{{remark.DATA}}`

内容示例

你好，你已刷卡成功。
银行卡号：尾号0449
消费金额：220元
时间：2014年7月21日 18:36
感谢你的使用。

*黑色字为实际填写部分

下一步

图 3-17

为了方便测试，开发者可以在测试公众号中新增测试模板。新增成功后，将会得到一个模板 ID，如图 3-18 所示。

模板消息接口				
新增测试模板 最多10个，接受模板消息需要关注测试号				
序号	模板ID(用于接口调用)	模板标题	模板内容	操作
1	N7miL0cTypMYAVbR mvHfxsJRbhmkkTTAm iEdh8VnELs	测试成功啦	<code>{{first.DATA}}</code> 测试时间： <code>{{keyword1.DATA}}</code> 测试奖励： <code>{{keyword2.DATA}}</code> <code>{{remark.DATA}}</code>	删除
2	nzrO9M_slAZ7YueyilD ct_TxztmYZI8qxgXdtu/ -xQI	新订单通知	<code>{{first.DATA}}</code> 提交时间： <code>{{tradeDateTime.DATA}}</code> 订单来源： <code>{{orderFrom.DATA}}</code> 客户信息： <code>{{customerInfo.DATA}}</code> 订单信息： <code>{{orderInfo.DATA}}</code> <code>{{remark.DATA}}</code>	删除

图 3-18



3.6.3 发送模板消息

发送模板消息的接口地址如下：

https://api.weixin.qq.com/cgi-bin/message/template/send?access_token=ACCESS_TOKEN

假如现在需要调用的模板内容是如图 3-18 所示的“新订单通知”模板，则需要 POST 的数据如代码清单 3-115 所示。

代码清单 3-115

```
{
  "touser": "o8mXItyyhhtlubxsO9nidRzfMIOU",
  "template_id": "nzm09M_sIAZ7YueyilDct_TxztmYZl8qXgXdtuJ-xQI",
  "url": "",
  "topcolor": "#cccccc",
  "data": {
    "first": {
      "value": "有一条新订单",
      "color": "#cc0000"
    },
    "tradeDateTime": {
      "value": "2015/2/25 23:45:36",
      "color": "#cc0000"
    },
    "orderFrom": {
      "value": "微信端",
      "color": "#cc0000"
    },
    "customerInfo": {
      "value": "张三 电话: 110",
      "color": "#cc0000"
    },
    "orderInfo": {
      "value": "凯迪拉克",
      "color": "#cc0000"
    },
    "remark": {
```



```
"value": "请及时确认",  
"color": "#cc0000"
```

根据代码清单 3-115 的数据格式和发送模板消息的接口可编写出发送模板消息的代码,如代码清单 3-116 所示。

代码清单 3-116

```
/// <summary>  
/// 发送模板消息  
/// </summary>  
/// <param name="touser">要发送的用户的 openid</param>  
/// <param name="template_id">模板 ID</param>  
/// <param name="topcolor">消息卡片顶部的颜色</param>  
/// <param name="dataKeys">模板字段列表</param>  
/// <param name="accessToken">accessToken</param>  
/// <param name="url">单击消息卡片跳转的地址。默认为空。如果为空, ios 设置会跳转到  
/// 空白页面; 安卓则不跳转</param>  
/// <returns>调用成功后, 返回的实体的 msgid 属性指的是此条模板消息的 ID</returns>  
public static TemplateMsg Send(string touser, string template_id, string  
topcolor, Dictionary<string, TemplateKey> dataKeys, string accessToken,  
string url = "")  
{  
    var turl =  
        string.Format("https://api.weixin.qq.com/cgi-bin/message/template/  
send?access_token={0}", accessToken);  
    var json = new  
    {  
        touser=touser,  
        template_id=template_id,  
        url=url,  
        topcolor=topcolor,  
        data=dataKeys  
    };  
    return Utils.PostResult<TemplateMsg>(json, turl);  
}
```



方法的返回类型 `TemplateMsg` 的定义如代码清单 3-117 所示。

代码清单 3-117

```
public class TemplateMsg : ErrorEntity
{
    public string msgid { get; set; }
}
```

另外，在方法中 `TemplateKey` 类封装了模板消息中每个属性的值与颜色，如下所示：

```
public class TemplateKey
{
    public string value { get; set; }
    public string color { get; set; }
}
```

下面的示例演示的是调用类 `TemplateNotice` 的 `Send` 方法，发送图 3-18 中模板标题为“新订单通知”的测试模板。在图 3-18 中可以看出，此模板包含 `first`、`tradeDateTime`、`orderFrom`、`customerInfo`、`orderInfo`、`remark` 关键词。

首先，在项目 `WxTest` 中添加一个测试页面 `TemplateTest`。在 `Page_Load` 方法中添加如代码清单 3-118 所示的代码。

代码清单 3-118

```
var accessToken = AccessTokenBox.GetTokenValue("wxd7008116979a800d",
    "49523ae424f3d22b369c16f8738fecba");
var dickKeys = new Dictionary<string, TemplateKey>();
dickKeys.Add("first", new TemplateKey { value = "有一条新订单", color =
    "#cc0000" });

dickKeys.Add("tradeDateTime", new TemplateKey { value = DateTime.Now.
    ToString(),
    color = "#cc0000" });
dickKeys.Add("orderFrom", new TemplateKey { value = "微信端",
    color = "#cc0000" });
dickKeys.Add("customerInfo", new TemplateKey { value = "张三 电话: 110",
    color = "#cc0000" });
dickKeys.Add("orderInfo", new TemplateKey { value = "凯迪拉克",
```



```
color = "#cc0000" });  
dicKeys.Add("remark", new TemplateKey { value = "请及时确认",  
color = "#cc0000" });  
var d=TemplateNotice.Send("o8mXItyyhhtlubxsO9nidRzfMIOU",  
"nzmR09M_sIAZ7YueyilDct_TxztmYZl8qXgXdtuJ-xQI", "#cccccc",  
dicKeys, accessToken);  
if (d.ErrCode==0)  
{  
    Response.Write("模板消息发送成功! 消息 ID 为: "+d.msgid);  
}  
else  
{  
    Response.Write("模板消息发送失败! 错误消息是: "+d.ErrDescription);  
}
```

然后打开此页面。页面加载完成后, 如果页面中输出的是“模板消息发送成功! 消息 ID 为……”, 则说明发送成功, 此时对应的 openid 的微信用户将收到如图 3-19 所示的消息。

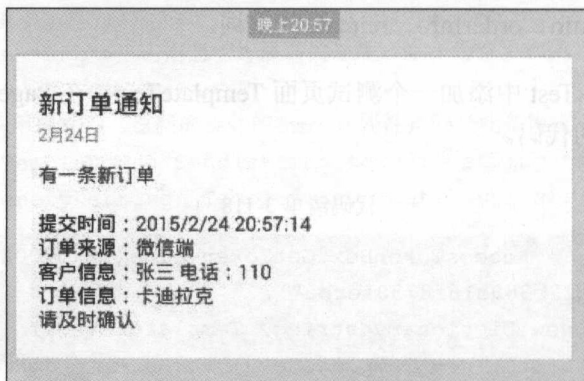


图 3-19

3.6.4 模板消息事件推送

和群发消息的事件推送一样, 在模板消息发送任务完成后, 微信服务器会将是否送达成功作为通知, 发送到开发者中心填写的服务器配置地址中。消息数据包如下所示。当送达成功时, Status 的值为 success。当由于用户拒收(用户设置拒绝接收公众号消息)而失败时 Status 的值为 failed:user block。由于其他原因失败时, Status 的值为 failed: system failed (见代码清单 3-119)。



代码清单 3-119

```
<xml>
  <ToUserName><![CDATA[gh_7f083739789a]]></ToUserName>
  <FromUserName><![CDATA[oia2TjuEGTNoeX76QEjQNrcURxG8]]></FromUserName>
  <CreateTime>1395658920</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[TEMPLATESENDJOBFINISH]]></Event>
  <MsgID>200163836</MsgID>
  <Status><![CDATA[success]]></Status>
</xml>
```

由于模板消息的事件推送和群发消息的事件推送非常类似，故此就不赘述了。读者可参考 3.5.4 节自行编写代码或参考实例代码。

3.7 推广支持

3.7.1 生成带参数的二维码

为了满足用户渠道推广分析的需要，公众平台提供了生成带参数二维码的接口。使用该接口可以获得多个带不同场景值的二维码，用户扫描后，公众号可以接收到事件推送。在推送的数据包中，包含了二维码的场景值。

目前有两种类型的二维码，分别是临时二维码和永久二维码。其中，前者有过期时间，最长可以设置为在二维码生成后的 7 天（即 604 800 秒）后过期，但能够生成较多数量；后者无过期时间，数量较少，目前最多能生成 10 万个。获取带参数的二维码的过程包括两步：首先创建二维码 ticket，然后凭借 ticket 到指定 URL 换取二维码；也可以在获取 ticket 后，调用第三方插件用接口返回的 URL 生成二维码。创建二维码 ticket 的接口地址如下：

https://api.weixin.qq.com/cgi-bin/qrcode/create?access_token=TOKEN

HTTP 请求方式：POST。

每次创建二维码 ticket 需要提供一个开发者自行设定的参数，临时二维码的场景 ID 的有效值为 32 位非 0 整型。永久二维码支持 1~100000 的数字和长度在 1~64 的字符串类型。



临时二维码 POST 的 JSON 格式如下:

```
{"expire_seconds": 1800, "action_name": "QR_SCENE", "action_info":  
{"scene": {"scene_id": 123}}}
```

永久二维码 POST 的 JSON 格式如下:

```
{"action_name": "QR_LIMIT_SCENE", "action_info": {"scene": {"scene_id":  
123}}}
```

或

```
{"action_name": "QR_LIMIT_STR_SCENE", "action_info": {"scene": {"scene_  
str": "123"}}}
```

在代码实现的过程中,首先在 WxApi 项目中创建类 QrCode,用于封装生成二维码相关的方法。根据接口地址在类 QrCode 中添加一个方法 Create (object obj, string accessToken),其中 obj 指的是 POST 的数据实体,accessToken 是接口调用凭据。由于两种二维码生成的接口地址是一样的,所以此方法只是封装了接口调用的部分,如代码清单 3-120 所示。

代码清单 3-120

```
private static QrTicket Create(object obj, string accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/cgi-bin/qrcode/create?access_  
token={0}", accessToken);  
    return Utils.PostResult<QrTicket>(obj, url);  
}
```

在上述代码清单中,返回值类型 QrTicket 是调用接口的返回信息对应的实体。具体代码如代码清单 3-121 所示。

代码清单 3-121

```
/// <summary>  
/// 创建二维码时,返回实体  
/// </summary>  
public class QrTicket:ErrorEntity  
{  
    /// <summary>  
    /// 获取的二维码 ticket,凭借此 ticket 可以在有效时间内换取二维码
```



```
/// </summary>
public string ticket { get; set; }
/// <summary>
/// 二维码的有效时间，以秒为单位。最大不超过 1800
/// </summary>
public int expire_seconds { get; set; }
/// <summary>
/// 二维码图片解析后的地址，开发者可根据该地址自行生成需要的二维码图片
/// </summary>
public string url { get; set; }
}
```

代码清单 3-122 中的代码是生成临时二维码和永久二维码 ticket 的方法。

代码清单 3-122

```
/// <summary>
/// 生成临时二维码
/// </summary>
/// <param name="expireSeconds">过期时间，最大不超过 604800，即 7 天</param>
/// <param name="sceneId">场景值 ID，临时二维码时为 32 位非 0 整型</param>
/// <param name="accessToken">accessToken</param>
/// <returns>ticket 实体，ticket 可以换取二维码，也可以根据 URL 自行生成。</returns>
public static QrTicket CreateTemp(int expireSeconds, int sceneId, string
accessToken)
{
    if (expireSeconds<=0||expireSeconds>1800)
    {
        return new QrTicket{ErrCode = -3,ErrDescription = "有效时间不合法"};
    }
    var json = new
    {
        expire_seconds = expireSeconds,
        action_name = "QR_SCENE",
        action_info = new {scene = new {scene_id = sceneId}}
    };
    return Create(json, accessToken);
}
/// <summary>
/// 创建场景值为数字的永久二维码
```



```
/// </summary>
/// <param name="sceneId">场景值, 有效范围 1-100000</param>
/// <param name="accessToken">accessToken</param>
/// <returns>ticket 实体, ticket 可以换取二维码, 也可以根据 URL 自行生成。</returns>
public static QrTicket CreateByID(int sceneId, string accessToken)
{
    if (sceneId < 1 || sceneId > 100000)
    {
        return new QrTicket { ErrCode = -3, ErrDescription = "场景值不合法, 有效范围 1-100000" };
    }
    var json = new
    {
        action_name = "QR_LIMIT_SCENE",
        action_info = new { scene = new { scene_id = sceneId } }
    };
    return Create(json, accessToken);
}

/// <summary>
/// 创建场景值为数字的永久二维码
/// </summary>
/// <param name="sceneId">场景值, 有效范围 1-100000</param>
/// <param name="accessToken">accessToken</param>
/// <returns>ticket 实体, ticket 可以换取二维码, 也可以根据 URL 自行生成。</returns>
public static QrTicket CreateByStr(string sceneStr, string accessToken)
{
    if (sceneStr.Length < 1 || sceneStr.Length > 64)
    {
        return new QrTicket { ErrCode = -3, ErrDescription = "场景值不合法, 长度限制为 1 到 64" };
    }
    var json = new
    {
        action_name = "QR_LIMIT_SCENE",
        action_info = new { scene = new { scene_str = sceneStr } }
    };
    return Create(json, accessToken);
}
```




在上面代码的方法中, 返回类型 `QrTicket` 的属性都是 `string` 类型, 看上去和二维码图片没有什么关系。但开发者将链接 `https://mp.weixin.qq.com/cgi-bin/showqrcode?ticket=TICKET` 中的 `ticket` 的值替换成上述方法获取到的真实 `ticket` 即可获得二维码的地址。此时, 就可以像显示普通图片一样来显示这个二维码。如在网页中显示应该这样写: ``。

代码清单 3-123 为通过 `ticket` 获取二维码地址的方法。

代码清单 3-123

```
public static string GetQrByTicket(string ticket)
{
    return string.Format("https://mp.weixin.qq.com/cgi-bin/showqrcode?ticket={0}", ticket);
}
```

如果需要将生成的二维码下载到客户端, 则可以调用 3.2.2 节中讲到的 `Utils` 类中的 `DownloadFile` 方法。

还有一种方法是调用第三方类库, 将获取 `ticket` 时返回的 URL 生成二维码, 扫描出的结果和使用 `ticket` 换取的二维码是一样的。`QrCode.Net` 是一个使用 C# 开发的用于生成二维码的开源类库。项目地址为: <http://qrcodenet.codeplex.com>。将文件下载之后解压, 可以看到如图 3-20 所示的文件列表。

名称	修改日
Gma.QrCodeNet.Encoder	2013/
Gma.QrCodeNet.Encoding.Net35	2013/
Gma.QrCodeNet.Encoding.Net45	2013/
Gma.QrCodeNet.Encoding.Silverlight	2013/
Gma.QrCodeNet.Encoding.WinRT	2013/

图 3-20

从图 3-20 中可以看出 `QrCodeNet` 支持 .NET 大部分框架, 可惜的是不支持 .NET 2.0。第一个文件的框架版本是 .NET 4.0。

在调用 `QrCodeNet` 前, 项目中添加对类库的引用, 然后在需要调用此类库的类中导入命名空间 `Gma.QrCodeNet.Encoding` 和 `Gma.QrCodeNet.Encoding.Windows.Render`。之后添加方法 `GetQrCodeStream`。此方法接收一个字符串参数表示二维码的内容, 以及一个数字



参数表示二维码的尺寸。生成二维码后，返回二维码内存流，如代码清单 3-124 所示。

代码清单 3-124

```
/// <summary>
/// 生成二维码流
/// </summary>
/// <param name="qrcontent">二维码的内容</param>
/// <param name="size">生成的二维码的尺寸。单位是像素</param>
/// <returns>内存流。可保存为文件，或下载到客户端</returns>
public static FileStreamInfo GetQrCodeStream(string qrcontent,int size)
{
    //误差校正水平
    ErrorCorrectionLevel ecLevel = ErrorCorrectionLevel.M;
    //空白区域
    QuietZoneModules quietZone = QuietZoneModules.Zero;
    int ModuleSize = size;//大小
    Gma.QrCodeNet.Encoding.QrCode qrCode;
    var encoder = new QrEncoder(ecLevel);
    //对内容进行编码，并保存生成的矩阵
    if (encoder.TryEncode(qrcontent, out qrCode))
    {
        var render = new GraphicsRenderer(new FixedCodeSize(ModuleSize,
quietZone));
        var stream = new FileStreamInfo();
        render.WriteToStream(qrCode.Matrix, ImageFormat.Jpeg, stream);
        stream.FileName = DateTime.Now.ToString("yyyyMMddHHmmssfff") + ".jpg";
        return stream;
    }
    return null;
}
```

调用上述方法后，在 B/S 架构中，如果需要将文件保存在本地，则可使用代码清单 3-125 所示的方式。

代码清单 3-125

```
Image img = Image.FromStream(stream);
img.Save(path);
```

其中，Image.FromStream(stream)中的 stream 就是调用 GetQrCodeStream 返回的



FileInfo 类型的对象, img.Save(path) 中的 path 表示的是文件要存放的物理路径。

3.7.2 扫描带参数二维码事件处理

用户扫描带场景值二维码时, 可能推送以下两种事件:

- 如果用户还未关注公众号, 则用户可以关注公众号, 关注后微信会将带场景值关注事件推送给开发者。
- 如果用户已经关注公众号, 则微信会将带场景值扫描事件推送给开发者。

若用户已关注公众号, 则扫描带参数二维码推送的 XML 数据包如代码清单 3-126 所示。

代码清单 3-126

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1425303137</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[SCAN]]></Event>
  <EventKey><![CDATA[1]]></EventKey>
  <Ticket><![CDATA[gQFO8ToAAAAAAAAASxodHRwOi8vd2VpeGluLnFxLmNvbS9xL2JVeE94TlBsakUtMVAYQUxBR0EyAAIEBOptVAMEAAAAAA==]]></Ticket>
</xml>
```

知道了数据包格式, 就可以根据 3.3 节中处理消息数据的方式来实现此事件的接口封装。首先, 根据 XML 数据包创建与之对应的实体, 如代码清单 3-127 所示。

代码清单 3-127

```
public class ScanQrEventMsg : EventMsg
{
    /// <summary>
    /// 事件 KEY 值
    /// </summary>
    public string EventKey { get; set; }
    /// <summary>
    /// 二维码的 ticket, 可用来换取二维码图片
    /// </summary>
```



```
public string Ticket { get; set; }  
}
```

然后，在类 `MsgFactory` 中，在处理事件类型的 `switch` 中添加如代码清单 3-128 所示的代码。

代码清单 3-128

```
case EventType.SCAN:  
    msg = Utils.ConvertObj<ScanQrEventMsg>(postStr);break;
```

最后再在 `WxTest` 项目的 `wx.ashx` 处理程序中，添加处理此事件的处理程序，并绑定到消息处理程序列表中。代码可参考 3.3 节中的实现方式，限于篇幅，不再赘述。

若用户未关注公众号，则扫描带参数二维码时，首先需要先关注公众号，关注后将推送如代码清单 3-129 所示的数据包。

代码清单 3-129

```
<xml>  
  <ToUserName><![CDATA[gh_8dlff4fd69cd]]></ToUserName>  
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>  
  <CreateTime>1425313514</CreateTime>  
  <MsgType><![CDATA[event]]></MsgType>  
  <Event><![CDATA[subscribe]]></Event>  
  <EventKey><![CDATA[qrscene_1]]></EventKey>  
  <Ticket><![CDATA[gQFO8ToAAAAAAAAAASxodHRwOi8vd2VpeGluLnFxLmNvbS9xL2JVeE94TlBsakUtMVAYQUxBR0EyAAIEBOptVAMEAAAAAA==]]></Ticket>  
</xml>
```

将上述数据包与 3.3 节中讲到的用户关注事件的数据包对比后会发现，扫描带参数二维码关注时推送的数据包比常规关注的数据包多了两个节点。一个是 `EventKey`，其值为 `qrscene_` 后面加上二维码的参数，所以在处理这个节点时，需将多余的前缀去掉。另一个字段是 `Ticket`，含义和关注时扫描一样。

3.7.3 长短链接转换接口

此接口的功能是将一条长链接转成短链接。主要使用场景：开发者用于生成二维码的原链接（商品、支付二维码等）太长，导致扫描速度和成功率下降，将原长链接通过此接



口转换成短链接再生成二维码将大大提升扫描速度和成功率。接口地址如下：

`https://api.weixin.qq.com/cgi-bin/shorturl?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

请求数据示例：`{"action ":"long2short","long_url ":"long_url"}`。

根据接口定义，在类 `BaseServices` 中添加方法 `LongUrlToShortUrl`。调用接口成功后，返回与返回信息对应的实体 `ShortUrl`，如代码清单 3-130 所示。

代码清单 3-130

```
/// <summary>
/// 将一条长链接转成短链接
/// </summary>
/// <param name="longurl">长链接</param>
/// <param name="accessToken">accessToken</param>
/// <returns>包含短链接和错误代码的实体</returns>
public static ShortUrl LongUrlToShortUrl(string longurl, string accessToken)
{
    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/shorturl?access_
token={0}", accessToken);
    var json = new { action = "long2short", long_url=longurl };
    return Utils.PostResult<ShortUrl>(json, url);
}
```

其中，返回类型 `ShortUrl` 的定义如代码清单 3-131 所示。

代码清单 3-131

```
public class ShortUrl:ErrorEntity
{
    public string short_url { get; set; }
}
```

3.8 自定义菜单

自定义菜单能够帮助公众号丰富界面，让用户更好、更快地理解公众号的功能。自定



义菜单的出现进一步模糊了公众号与 App 的界限。目前自定义菜单最多可包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单。一级菜单最多为 4 个汉字，二级菜单最多为 7 个汉字，多出来的部分将会以 “...” 代替。需要注意的是，创建自定义菜单后，可能会由于客户端缓存，而不会立即展现出来。建议测试时可以尝试取消关注公众账号后再次关注，这时可以看到创建后的效果。

自定义菜单接口可实现多种类型按钮，如表 3-5 所示。

表 3-5

类 型	说 明	单击按钮后的响应
click	单击推事件	用户单击按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给开发者（参考消息接口指南），并且带上按钮中开发者填写的 key 值。开发者可以通过自定义的 key 值与用户进行交互
view	跳转 URL	用户单击按钮后，微信客户端将会打开开发者在按钮中填写的网页 URL，可与网页授权获取用户基本信息接口结合，获得用户基本信息
scancode_push	扫码推事件	用户单击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后显示扫描结果（如果是 URL，则将进入 URL），且会将扫码的结果传给开发者，开发者可以下发消息
scancode_waitmsg	扫码推事件且弹出“消息接收中”提示框	用户单击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后，将扫码的结果传给开发者，同时收起扫一扫工具，然后弹出“消息接收中”提示框，随后可能会收到开发者下发的消息
pic_sysphoto	弹出系统拍照发图	用户单击按钮后，微信客户端将调起系统相机，完成拍照操作后，会将拍摄的相片发送给开发者，并推送事件给开发者，同时收起系统相机，随后可能会收到开发者下发的消息
pic_photo_or_album	弹出拍照或者相册发图	用户单击按钮后，微信客户端将弹出选择器供用户选择“拍照”或者“从手机相册选择”。用户选择后即走其他两种流程
pic_weixin	弹出微信相册发图器	用户单击按钮后，微信客户端将调起微信相册，完成选择操作后，将选择的相片发送给开发者的服务器，并推送事件给开发者，同时收起相册，随后可能会收到开发者下发的消息
location_select	弹出地理位置选择器	用户单击按钮后，微信客户端将调起地理位置选择工具，完成选择操作后，将选择的地理位置发送给开发者的服务器，同时收起位置选择工具，随后可能会收到开发者下发的消息

3.8.1 自定义菜单创建接口

创建自定义菜单的接口地址如下：

https://api.weixin.qq.com/cgi-bin/menu/create?access_token=ACCESS_TOKEN



HTTP 请求方式: POST。

在调用时,只需将相应的 JSON 字符串 POST 到上述的接口地址。如果返回的数据包如{"errcode":0,"errmsg":"ok"}所示,则表示创建成功。

POST 的 JSON 字符串如代码清单 3-132 所示。

代码清单 3-132

```
{
  "button": [
    {
      "type": "click",
      "name": "今日歌曲",
      "key": "V1001_TODAY_MUSIC"
    },
    {
      "name": "菜单",
      "sub_button": [
        {
          "type": "view",
          "name": "搜索",
          "url": "http://www.soso.com/"
        },
        {
          "type": "view",
          "name": "视频",
          "url": "http://v.qq.com/"
        }
      ]
    }
  ]
}
```

在上述清单中, button 表示的是一级菜单数组,个数应为 1~3 个。sub_button 是可选的,表示的是二级菜单数组,个数应为 1~5 个。type 则表示的是菜单的响应动作类型,即本节开篇讲到的类型列表。name 是菜单标题,不超过 16 个字节;子菜单不超过 40 个字节。url 是 view 类型特有的字段,表示的是单击 view 类型的菜单跳转的地址。key 则表示的是菜单 key 值,是除了 view 类型之外的菜单所必需的,用于消息接口推送,不超过 128 字节。



为了方便调用，可以把各个菜单类型封装成枚举，如代码清单 3-133 所示。

代码清单 3-133

```
namespace WxApi
{
    public enum MenuType
    {
        click,
        view,
        scancode_push,
        scancode_waitmsg,
        pic_sysphoto,
        pic_photo_or_album,
        pic_weixin,
        location_select
    }
}
```

然后根据 JSON 数据包创建类 BaseMenu，映射的是基础菜单信息，包括菜单标题、类型、子菜单等。创建类 MenuEntity，此类有一个属性 button，类型是 BaseMenu 列表，如代码清单 3-134 所示。

代码清单 3-134

```
namespace WxApi.SendEntity
{
    /// <summary>
    /// 基础菜单信息
    /// </summary>
    public class BaseMenu
    {
        /// <summary>
        /// 菜单的响应动作类型
        /// </summary>
        public MenuType type { get; set; }
        /// <summary>
        /// 菜单标题，不超过 16 个字节；子菜单不超过 40 个字节
        /// </summary>
    }
}
```




```

public string name { get; set; }
/// <summary>
/// click 等单击类型必填 菜单 key 值, 用于消息接口推送, 不超过 128 字节
/// </summary>
public string key { get; set; }
/// <summary>
/// view 类型必填 网页链接, 用户单击菜单可打开链接, 不超过 256 字节
/// </summary>
public string url { get; set; }
/// <summary>
/// 子菜单
/// </summary>
public List<BaseMenu> sub_button { get; set; }

}
/// <summary>
/// 自定义菜单实体
/// </summary>
public class MenuEntity
{
    public List<BaseMenu> button { get; set; }
}
}

```

最后在项目中创建类 `Menu`, 用于封装自定义菜单相关操作的方法。在类中添加方法 `Create`, 此方法接收两个参数: 一个类型为 `MenuEntity` 的 `menuEntity` 和一个字符串类型的 `accessToken`, 返回值类型是 `ErrorEntity`, 如代码清单 3-135 所示。

代码清单 3-135

```

public static ErrorEntity Create(MenuEntity menuEntity, string accessToken)
{
    var url =
        string.Format(" https://api.weixin.qq.com/cgi-bin/menu/create?
access_token={0}", accessToken);
    return Utils.PostResult<ErrorEntity>(menuEntity, url);
}

```

代码清单 3-136 是调用 `Create` 方法创建自定义菜单的实例代码, 代码中实例化了 3 个



菜单基类，分别在每个基类中添加了各个分类的子菜单。

代码清单 3-136

```
var accessToken = AccessTokenBox.GetTokenValue(appid, appsecret);
var child1 = new List<BaseMenu>();
var child2 = new List<BaseMenu>();
var child3 = new List<BaseMenu>();
var basebtn = new List<BaseMenu>();
child1.Add(new BaseMenu
{
    key = "我是 click 按钮",
    name = "Click 按钮",
    type = MenuType.click
});
child1.Add(new BaseMenu
{
    key = "我是选择地理位置按钮",
    name = "选择地理位置",
    type = MenuType.location_select
});
child1.Add(new BaseMenu
{
    url = "http://www.baidu.com",
    name = "跳转链接",
    type = MenuType.view
});
child2.Add(new BaseMenu
{
    key = "我是扫码推事件按钮",
    name = "扫码推事件",
    type = MenuType.scancode_push
});
child2.Add(new BaseMenu
{
    key = "我是扫码推事件按钮且弹出消息接收中",
    name = "扫码等待",
    type = MenuType.scancode_waitmsg
});
```



```
});

child3.Add(new BaseMenu
{
    key = "我是拍照或相册按钮",
    name = "拍照或相册",
    type = MenuType.pic_photo_or_album
});
child3.Add(new BaseMenu
{
    key = "我是系统拍照",
    name = "系统拍照",
    type = MenuType.pic_sysphoto
});
child3.Add(new BaseMenu
{
    key = "我是弹出微信相册按钮",
    name = "微信相册",
    type = MenuType.pic_weixin
});

basebtn.Add(new BaseMenu
{
    name = "常用菜单",
    sub_button = child1
});
basebtn.Add(new BaseMenu
{
    name = "扫码",
    sub_button = child2
});
basebtn.Add(new BaseMenu
{
    name = "发图",
    sub_button = child3
});

var ret = WxApi.Menu.Create(new MenuEntity { button = basebtn }, accessToken);
```



执行上述代码后，生成的自定义菜单如图 3-21 所示。

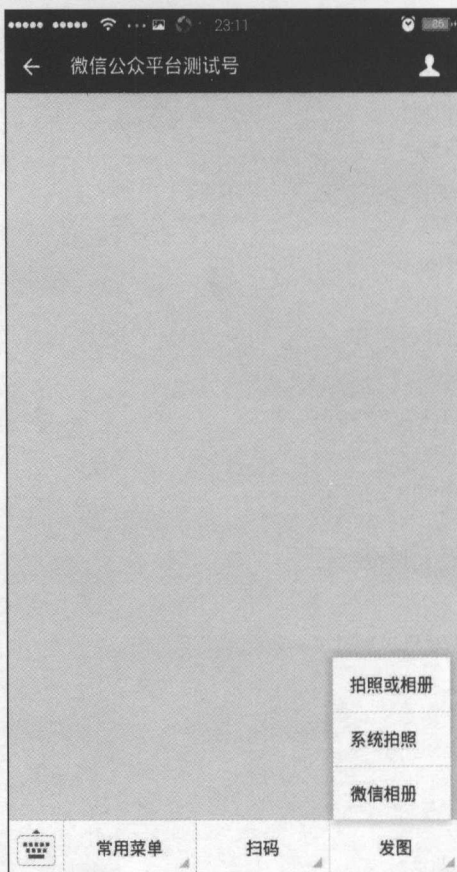


图 3-21

3.8.2 自定义菜单查询接口

使用接口创建自定义菜单后，开发者还可以使用接口查询自定义菜单的结构。接口的 URL 如下：

`https://api.weixin.qq.com/cgi-bin/menu/get?access_token=ACCESS_TOKEN`

HTTP 请求方式：GET。

返回的结果和创建接口时提交的 json 对应，只不过是属于 menu 对象的子属性。实现



代码如代码清单 3-137 所示。

代码清单 3-137

```
public static MenuEntity Query(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/menu/get?access_
token={0}", accessToken);
    var job = Utils.GetResult<JsonObject>(url);
    var menu = job["menu"].ToString();
    return JsonConvert.DeserializeObject<MenuEntity>(menu);
}
```

3.8.3 自定义菜单删除接口

自定义菜单创建以后，是一直有效的，开发者还可以使用接口删除当前使用的自定义菜单。删除菜单的接口地址如下：

https://api.weixin.qq.com/cgi-bin/menu/delete?access_token=ACCESS_TOKEN

HTTP 请求方式：GET。

对应的代码如代码清单 3-138 所示。

代码清单 3-138

```
public static ErrorEntity Delete(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/menu/delete?access_
token={0}", accessToken);
    return Utils.GetResult<ErrorEntity>(url);
}
```

3.8.4 自定义菜单事件推送

用户单击自定义菜单后，微信会把单击事件推送给开发者。请注意，如果一级菜单有子菜单，则单击时不会推送任何信息。在这里，以 3.7.1 节中生成的菜单为例，看一下单击每个按钮之后推送的消息内容（见代码清单 3-139）。



执行上述代码后，生成代码清单 3-139（“常用菜单”“Click 按钮”）

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1426779984</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[CLICK]]></Event>
  <EventKey><![CDATA[我是click按钮]]></EventKey>
</xml>
```

click 类型的按钮在被单击时，将推送如上述清单所示的 XML 数据包。其中 EventKey 为创建按钮时，数据包里的 key 值。除 view 类型按钮外，其他类型的 EventKey 的值，也都和创建按钮时数据包的 key 值是一致的（见代码清单 3-140）。

代码清单 3-140（“常用菜单”“跳转链接”）

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1426780155</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[VIEW]]></Event>
  <EventKey><![CDATA[http://www.baidu.com]]></EventKey>
</xml>
```

view 类型的按钮在被单击时，在跳转 URL 的同时，推送上述 XML 数据包给开发者的服务器。其中 EventKey 为创建菜单时，数据包里的 URL（见代码清单 3-141）。

代码清单 3-141（“常用菜单”“选择地理位置”）

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1426781682</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[location_select]]></Event>
  <EventKey><![CDATA[我是选择地理位置按钮]]></EventKey>
  <SendLocationInfo>
    <Location_X><![CDATA[30.52884292602539]]></Location_X>
    <Location_Y><![CDATA[114.34750366210938]]></Location_Y>
    <Scale><![CDATA[16]]></Scale>
```



```
<Label><![CDATA[武汉市武昌区武珞路 689 号]]></Label>
<Poiname><![CDATA[]]></Poiname>
</SendLocationInfo>
</xml>
```

location 类型的按钮在被单击时, 首先弹出如图 3-22 所示的地图选择器。选择地理位置后, 单击“发送”按钮, 将推送上述清单的 XML 数据包和 3.3.2 节中讲到的发送地理位置信息的数据包两种数据包。在上述数据包中, 将地理位置相关的信息作为节点 SendLocationInfo 的子节点。其中 Poiname 表示的是朋友圈 POI 的名字, 可能为空。其他字段则和发送地理位置的数据包类似, 在此不一一赘述。



图 3-22

scancode_push 类型的菜单在被单击时, 首先弹出扫一扫的页面。当扫描二维码后, 微信客户端会做两件事。一件事是推送上述 XML 结构的数据包给开发者的服务器。在下面所示的代码清单中 (见代码清单 3-142), ScanCodeInfo 即二维码的信息。其中 ScanType 是



二维码的类型, ScanResult 二维码的结果。另一件事就是直接处理用户的扫描结果。比如说, 用户扫的二维码的内容是链接, 则扫描之后会直接跳转到该链接。而如果扫描的是 MECARD 格式的名片二维码 (MECARD:TEL:18000000000; URL:http://qq.com;EMAIL: 000@qq.com;NOTE:123456789;N:张三;ORG:腾讯总公司;TIL:CEO;ADR:广东;), 则扫描后的结果如图 3-23 所示。

代码清单 3-142 (“扫码”“扫码推事件”)

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1426922227</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[scancode_push]]></Event>
  <EventKey><![CDATA[我是扫码推事件按钮]]></EventKey>
  <ScanCodeInfo>
    <ScanType><![CDATA[qrcode]]></ScanType>
    <ScanResult><![CDATA[http://baidu.com]]></ScanResult>
  </ScanCodeInfo>
</xml>
```



图 3-23



scancode_waitmsg 类型的菜单和 scancode_push 类型的菜单在扫描二维码后,推送给开发者服务器的数据包格式是一样的,但处理上是有本质区别的。前者在扫描之后只是将扫描的结果推送给开发者的服务器,至于做什么响应,完全由开发者自己处理;而后者首先由微信客户端做出处理,然后开发者也可以根据扫描的结果进行进一步的处理。

发送图片的菜单有三种类型。第一种是 pic_sysphoto (弹出系统拍照发图),也就是启动系统照相机拍照后发送。第二种是 pic_weixin (弹出微信相册发图器)。第三种是 pic_photo_or_album (弹出拍照或者相册发图的事件推送),是前两种的结合,当单击此菜单时,会弹出如图 3-24 所示的选项。三种类型的发图菜单在选择图片并单击“发送”按钮后,微信做的处理都是一样的,首先推送上述代码清单所示的 XML 数据包(见代码清单 3-143),其次分别将用户选择的图片以发送普通图片消息的形式推送给开发者的服务器(请参照 3.3.1 节)。在上述代码清单中,节点 SendPicsInfo 表示的是用户发送的图片信息,子节点 Count 表示的是发送的图片数量。子节点 PicList 中是一个列表,其中的 PicMd5Sum 表示的是图片的 MD5 值,开发者若需要,可用于验证接收到的图片。

代码清单 3-143 (“发图”“微信相册”)

```
<xml>
  <ToUserName><![CDATA[gh_8d1ff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1426939517</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[pic_weixin]]></Event>
  <EventKey><![CDATA[我是弹出微信相册按钮]]></EventKey>
  <SendPicsInfo>
    <Count>2</Count>
    <PicList>
      <item>
        <PicMd5Sum><![CDATA[61212b0a296c71c367635ea79c8942de]]></PicMd5Sum>
      </item>
      <item>
        <PicMd5Sum><![CDATA[5316a26102b6a541a07958f88a782084]]></PicMd5Sum>
      </item>
    </PicList>
  </SendPicsInfo>
</xml>
```

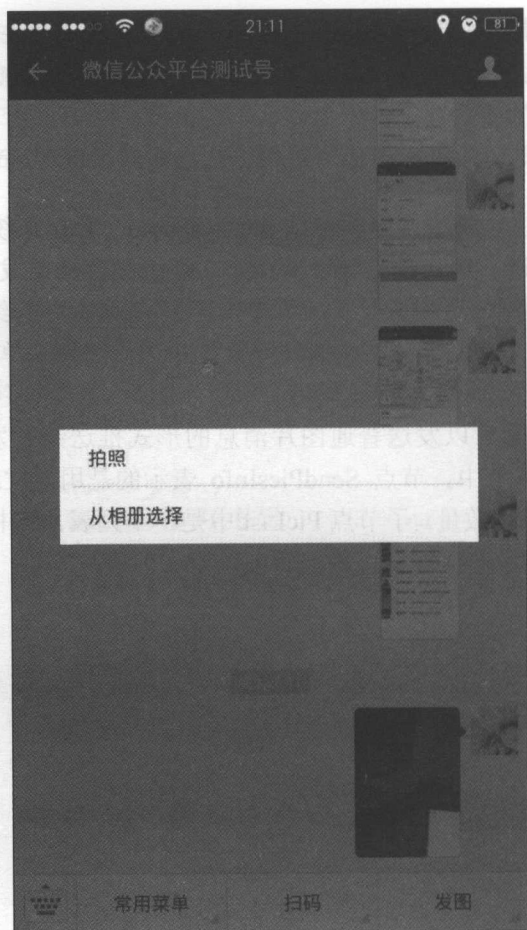


图 3-24

各个菜单的 XML 数据格式已经知道了，那下面就是根据 XML 来封装对应的实体映射了。因为所有菜单推送的 XML 都有 EventKey 属性，所以创建一个类 BaseMenuEventMsg。此类继承类 EventMsg，有一个属性 EventKey，如代码清单 3-144 所示。

代码清单 3-144

```
/// <summary>
/// 菜单事件基础实体
/// </summary>
public class BaseMenuEventMsg:EventMsg
{
```



```
/// <summary>
/// 事件 key 值
/// </summary>
public string EventKey { get; set; }
}
```

BaseMenuEventArgs 类已经包含了 click 和 view 类型菜单所推送的 XML 的所有属性，所以此类也可以作为 click 和 view 的实体映射。

从扫描二维码后推送的 XML 数据包中可以看出，二维码的扫描结果是体现在 ScanCodeInfo 节点的子节点中的。为了方便后期程序的调用和可读性，此处设计实体映射的时候可将子节点 ScanType 和 ScanResult 作为实体的直接属性进行处理，如代码清单 3-145 所示。

代码清单 3-145

```
public class ScanMenuEventArgs : BaseMenuEventArgs
{
    /// <summary>
    /// 扫描类型，一般是 qrcode
    /// </summary>
    public string ScanType { get; set; }
    /// <summary>
    /// 扫描结果，即二维码对应的字符串信息
    /// </summary>
    public string ScanResult { get; set; }
}
```

同理，在自定义菜单发送图片推送的 XML 数据包中，和图片相关的属性也是作为子节点存在的，在做实体映射的时候也将响应属性作为实体类的直接属性进行处理，如代码清单 3-146 所示。

代码清单 3-146

```
public class PicMenuEventArgs : BaseMenuEventArgs
{
    /// <summary>
    /// 发送的图片数量
    /// </summary>
}
```



```
public int Count { get; set; }  
/// <summary>  
/// 图片的 MD5 集合  
/// </summary>  
public List<string> PicMd5SumList { get; set; }  
}
```

消息实体创建完毕后,又有一个问题需要解决。因为在 3.3.3 节(消息数据包解析)中,首先是给共用属性赋值,其次再遍历接收到的 XML 节点,给消息体特有的属性赋值;而在遍历的代码中并没有针对有子节点的 XML 节点进行特殊处理,所以就要求消息实体要和接收到的 XML 一一对应。然而,在上述代码清单中,为了方便调用以及提高代码的可读性,我们已经对实体进行了调整,那么 3.3.3 节中讲到的用于处理消息数据包与实体之间解析的方法 `ConvertObj<T>(string xmlstr)` 就需要做相应的调整才可以。其实解决方法也很简单,我们只需在遍历的过程中判断是否是菜单扫描二维码事件或菜单发送图片事件,然后再做响应的处理即可。方法 `ConvertObj<T>(string xmlstr)` 最终的代码如代码清单 3-147 所示。

代码清单 3-147

```
public static T ConvertObj<T>(string xmlstr)  
{  
    try  
    {  
        XElement xdoc = XElement.Parse(xmlstr);  
        //获取转换的数据类型  
        var type = typeof(T);  
        //创建实例  
        var t = Activator.CreateInstance<T>();  
        #region 基础属性赋值  
        var ToUserName = type.GetProperty("ToUserName");  
        ToUserName.SetValue(t,  
Convert.ChangeType(xdoc.Element("ToUserName").Value,  
ToUserName.PropertyType), null);  
        xdoc.Element("ToUserName").Remove();  
  
        var FromUserName = type.GetProperty("FromUserName");  
        FromUserName.SetValue(t,  
Convert.ChangeType(xdoc.Element("FromUserName").Value,  
FromUserName.PropertyType), null);
```




```
xdoc.Element("FromUserName").Remove();

var CreateTime = type.GetProperty("CreateTime");
CreateTime.SetValue(t,
Convert.ChangeType(xdoc.Element("CreateTime").Value,
CreateTime.PropertyType), null);
xdoc.Element("CreateTime").Remove();

var MsgType = type.GetProperty("MsgType");
string msgtype = xdoc.Element("MsgType").Value.ToUpper();
MsgType.SetValue(t,
(MsgType)Enum.Parse(typeof(MsgType), msgtype), null);
xdoc.Element("MsgType").Remove();

//判断消息类型是否是事件
if (msgtype == "EVENT")
{
    //获取事件类型
    var EventType = type.GetProperty("Event");
    string eventtype = xdoc.Element("Event").Value.ToUpper();
    EventType.SetValue(t,
(EventType)Enum.Parse(typeof(EventType), eventtype), null);
    xdoc.Element("Event").Remove();
}
#endregion

//遍历 XML 节点
foreach (XElement element in xdoc.Elements())
{
    if (msgtype == "EVENT")
    {
        if (element.Name == "ScanCodeInfo")
        {
            type.GetProperty("ScanType").SetValue(t,
Convert.ChangeType(element.Element("ScanType").Value, TypeCode.String), null);
            type.GetProperty("ScanResult").SetValue(t,
Convert.ChangeType(element.Element("ScanResult").Value, TypeCode.String),
null);

            continue;
        }
    }
}
```



```
        }
        if (element.Name == "SendPicsInfo")
        {
            type.GetProperty("Count").SetValue(t,
Convert.ChangeType(element.Element("Count").Value, TypeCode.Int32), null);
            List<string> picMd5List = new List<string>();
            foreach (XElement xElement in element.Element("PicList").
Elements())
            {
                picMd5List.Add(xElement.Element("PicMd5Sum").Value);
            }
            type.GetProperty("PicMd5SumList").SetValue(t,
picMd5List, null);
            continue;
        }
    }
    //根据 XML 节点的名称, 获取实体的属性
    var pr = type.GetProperty(element.Name.ToString());
    //给属性赋值
    pr.SetValue(t, Convert.ChangeType(element.Value,
pr.PropertyType), null);
}
return t;
}
catch (Exception)
{
    return default(T);
}
}
```

最后, 还需要在我们的测试项目的 wx.ashx 的处理程序中, 按照 3.3.4 节 (消息处理页面示例) 所讲的, 将回调函数进行绑定。代码实现可参考 3.3.4 节, 在此不再赘述。

3.9 消息体签名及加解密

出于安全考虑, 微信公众平台在 2014 年 10 月加入了消息体的加解密功能。首先, 需要先验证签名, 用于公众平台和公众账号验证消息体的正确性。然后, 针对推送给公众号



的普通消息和事件消息，以及推送给设备公众账号的设备进行加密。最后，公众号对密文消息的回复也需要加密。启动加解密功能后，公众平台服务器在向开发者服务器推送消息时，URL 将新增加两个参数：一个是加密类型，另一个是消息体签名，并以此来体现新功能。

3.9.1 加解密模式介绍

为了配合消息加密功能的上线，并帮助开发者适配新特性，公众平台提供了 3 种加解密的模式供开发者选择，即明文模式、兼容模式、安全模式（可在“开发者中心”选择相应的模式）。选择兼容模式和安全模式前，需在开发者中心填写消息加解密密钥 EncodingAESKey。

明文模式：公众号接入的最原始模式。没有适配加解密特性，消息体明文收发。默认设置为明文模式。

兼容模式：公众平台推送给开发者服务器的消息数据包包括明文和密文，消息包长度增加到原来的 3 倍左右，公众号回复明文或密文均可，不影响现有消息收发，开发者可在此模式下进行调试。正式运营下，不建议使用兼容模式。

安全模式（推荐）：公众平台推送的消息体的内容只包含密文，进行消息回复时，消息体也为密文。建议开发者在正式运营时，使用此模式。

3.9.2 接入指南

为了方便开发者，微信官方提供了加解密相关的方法。这使得开发者在接入加解密方式时将变得非常简单。在接入之前，首先下载官方提供的类库，下载地址是：
<http://mp.weixin.qq.com/wiki/static/assets/a5a22f38cb60228cb32ab61d9e4c414b.zip>。

将下载的文件解压后，将其中的 Cryptography.cs 和 WXBizMsgCrypt.cs 添加到项目中。类 WXBizMsgCrypt 的构造函数包含 3 个参数。sToken 表示的是公众平台上，开发者设置的 Token；sEncodingAESKey 表示用于加解密的密钥，在公众平台设置；sAppID 是公众号的 AppID，唯一标识一个公众号。在调用加解密函数之前，需调用此构造函数进行初始化。

DecryptMsg 方法的功能是检验消息的真实性，并且在确认消息真实时，获取解密后的明文。此方法的参数说明如表 3-6 所示。



表 3-6

参 数 名	类 型	说 明
sMsgSignature	string	签名串, 对应 URL 参数的 msg_signature
sTimeStamp	string	时间戳, 对应 URL 参数的 timestamp
sNonce	string	随机串, 对应 URL 参数的 nonce
sPostData	string	密文, 对应 POST 请求的数据
sMsg	string	解密后的原文。当 return 返回 0 时有效

EncryptMsg 方法的功能是将开发者回复给用户的消息进行加密处理。此方法在回复消息前使用。参数说明如表 3-7 所示。

表 3-7

参 数 名	类 型	说 明
sReplyMsg	string	待回复的 XML 字符串, 即在接入加解密功能前的原始字符串
sTimeStamp	string	时间戳, 可以自己生成, 也可以用 URL 参数的 timestamp
sNonce	string	随机串, 可以自己生成, 也可以用 URL 参数的 nonce
sEncryptMsg	string	加密后的可以直接回复用户的密文, 包括 msg_signature、timestamp、nonce、encrypt 的 XML 格式的字符串。当 return 返回 0 时有效

在安全模式或兼容模式下, URL 上会新增两个参数 encrypt_type 和 msg_signature。encrypt_type 表示加密类型, msg_signature 表示对消息体的签名。URL 上无 encrypt_type 参数或者其值为 raw 时表示为不加密; encrypt_type 为 aes 时, 表示 aes 加密(暂时只有 raw 和 aes 两种值)。公众账号开发者根据此参数来判断微信公众平台发送的消息是否加密。

由于在调用加密和解密的方法时都需要将类 WXBizMsgCrypt 进行实例化, 因此为了方便调用, 将此类的构造函数所需要的参数封装成实体, 如代码清单 3-148 所示。

代码清单 3-148

```
/// <summary>
/// 微信接入参数
/// </summary>
public class EnterParam
{
    /// <summary>
    /// 是否加密
    /// </summary>
    public bool IsAes { get; set; }
```




```

/// <summary>
/// 接入 token
/// </summary>
public string token { get; set; }
/// <summary>
/// 微信 AppID
/// </summary>
public string appid { get; set; }
/// <summary>
/// 加密密钥
/// </summary>
public string EncodingAESKey { get; set; }
}

```

类 EnterParam 的参数 IsAes 表示的是是否加密。当用户接收到用户请求时, 判断是否加密; 当需要回复消息时, 根据 IsAes 判断是否需要将回复用户的消息体进行加密。解析微信服务器推送的 XML 的方法已经在 3.3.3 节中进行了详细的讲解。在接入加解密时, 只需稍微修改一下接收 XML 数据的处理方法。在 3.3.3 节中讲到的 MsgFactory 类的 LoadMsg 方法中, 使用 Utils.GetRequestData() 来获取微信服务器发送过来的数据, 如图 3-25 所示。

```

//获取数据包
var postStr = Utils.GetRequestData();
XElement xdoc = XElement.Parse(postStr);
var msgtype = xdoc.Element("MsgType").Value.ToUpper();
var FromUserName = xdoc.Element("FromUserName").Value;
var CreateTime = xdoc.Element("CreateTime").Value;
//返回消息类型

```

图 3-25

现在, 为了接入加解密方案, 在不大改原有代码的基础上, 唯一能改动的就是 Utils.GetRequestData() 方法了。我们只需在获取到微信服务器发送过来的原始数据后, 根据 URL 的查询字符串 encrypt_type 来判断是否进行了加密。如果没有加密, 则直接返回原始数据; 如果加密了, 则加密成功后, 返回真实的明文数据。具体实现如代码清单 3-149 所示。

代码清单 3-149

```

/// <summary>
/// 获取微信发送过来的数据包, 并处理加解密问题。如果进行了加密, 则将密文解密后返回; 否
/// 则直接返回接收到的字符

```



```
/// </summary>
/// <param name="param">加解密接入参数</param>
/// <returns>如果进行了加密,则将密文解密后返回;否则直接返回接收到的字符</returns>
public static string GetRequestData(EnterParam param)
{
    //获取当前请求的原始数据包
    var reqdata = GetRequestData();
    var timestamp = HttpContext.Current.Request.QueryString["timestamp"];
    var nonce = HttpContext.Current.Request.QueryString["nonce"];
    var msg_signature =
    HttpContext.Current.Request.QueryString["msg_signature"];
    var encrypt_type = HttpContext.Current.Request.QueryString["encrypt_
    type"];
    string postStr = null;
    if (encrypt_type == "aes")
    {
        //如果进行了加密,则加密成功后,直接返回解密后的明文。解密失败则返回 null
        param.IsAes = true;
        var ret = new WXBizMsgCrypt(param.token, param.EncodingAESKey,
        param.appid);
        if (ret.DecryptMsg(msg_signature, timestamp, nonce, reqdata, ref
        postStr) != 0)
        {
            return null;
        }
        return postStr;
    }
    else
    {
        param.IsAes = false;
        return reqdata;
    }
}
```

上述代码接收一个类型为 EnterParam 的参数,此参数是和加解密相关的参数,所以类 MsgFactory 的方法 LoadMsg 也需要增加一个这样的参数,修改后的代码如图 3-26 所示。



```

1 15188
public static void LoadMsg(EnterParam param, List<MsgHandlerEntity> mheList)
{
    //判断队列是否为空，为空则实例化。
    if (_queue == null)
    {
        _queue = new List<BMMsg>();
    }
    else
    {
        //保留20秒内未响应的消息
        _queue = _queue.Where(q => q.CreateTime.AddSeconds(20) > DateTime.Now).ToList();
    }

    //获取数据报文
    var postStr = Utils.GetRequestData(param);
    XElement xdoc = XElement.Parse(postStr);

```

图 3-26

最后在项目“WxTest”的 wx.ashx 处理页面中，添加类型为 EnterParam 的静态变量，在 ProcessRequest 方法的开头对此静态变量赋值。将 param 参数传入到方法 MsgFactory.LoadMsg(EnterParam param)中，如图 3-27 所示。

```

0 个引用
public class wx : IHttpHandler
{
    static EnterParam param;
    0 个引用
    public void ProcessRequest(HttpContext context)
    {
        //如果param为null，则实例化
        param = param ?? new EnterParam
        {
            appid = "wx8effbfd02ac34114",
            EncodingAESKey = "dD26XtlxqJrURMDC3pmITK9tqjGE2vx3oLq3ohxofL",
            token = "qqq"
        };
    }
}

```

图 3-27

同理，在回复消息给用户前，首先判断公众号是否接入了加解密方案。如果是，则调用官方提供的加密方法，将明文进行加密后，再将密文回复给用户。如果否，则直接进行回复。在 BaseMsg 类中添加如代码清单 3-150 所示的方法。

代码清单 3-150

```

private void Response(EnterParam param, string data)
{
    if (param.IsAes)

```



```
{  
    var wxcpt = new WXBizMsgCrypt(param.token, param.EncodingAESKey,  
    param.appid);  
    wxcpt.EncryptMsg(data, Utils.ConvertDateTimeInt(DateTime.Now).  
    ToString(), Utils.ConvertDateTimeInt(DateTime.Now).ToString(), ref data);  
}  
HttpContext.Current.Response.Write(data);  
HttpContext.Current.Response.End();  
}
```

最后将 3.4.1 节（被动响应消息）中的回复用户消息的代码按照表 3-8 中的方式进行修改即可。

表 3-8

未接入加解密	接入加解密
<pre>public virtual void ResText(string content) { var resxml = new StringBuilder(); //此处省略拼接 XML 字符串的代码，具体可参考 3.3.1 节 //中的代码 HttpContext.Current.Response.Write(resxml); return; }</pre>	<pre>public virtual void ResText(EnterParam param, string content) { var resxml = new StringBuilder(); //此处省略拼接 XML 字符串的代码，具体可参考 //3.3.1 节中的代码 //代码清单 3-135 中的方法 Response(param, resxml.ToString()); }</pre>

第4章 订阅用户管理

微信公众号的管理很大一部分工作量都是围绕着“订阅用户”来展开的。但在微信官方提供的管理后台中，用户只是简单地列起来，没有查找，没有更多条件的筛选，让公众号的运营者管理起来确实有点“捉急”。然而，用户管理相关接口让开发者和运营者看到了些许曙光。我们可以调用接口获取用户的详细信息，包括昵称、头像、性别、地区等。我们也可以备注特殊的用户，可以使用网页授权在我们的微网站里使用微信用户信息。

4.1 分组管理接口

用户分组管理，是在用户管理中最行之有效的办法。开发者可以使用此接口，对公众平台的用户分组进行查询、创建、修改操作，也可以使用接口在需要时移动用户到某个分组。

4.1.1 创建分组

目前，一个公众号最多支持创建 100 个分组。创建分组的接口地址如下：

https://api.weixin.qq.com/cgi-bin/groups/create?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例：`{"group":{"name":"test"}}`。



正常情况下，返回的 JSON 数据包如代码清单 4-1 所示。

代码清单 4-1

```
{
  "group": {
    "id": 107,
    "name": "test"
  }
}
```

说句题外话，其实蛮想“吐槽”一下微信的工程师：我只是创建一个分组，创建成功返回一个 id 不就行了吗？像上述的 JSON 数据包着实是多此一举，在创建与返回值对应的实体，以及调用时都不是很方便。哎……“吐槽”归“吐槽”，还是要按照接口来编写代码了。为了方便调用，这里在创建返回实体时进行了简化，如代码清单 4-2 所示。

代码清单 4-2

```
/// <summary>
/// 用户分组实体
/// </summary>
public class UserGroupEntity: BaseEntity
{
    /// <summary>
    /// 分组 ID
    /// </summary>
    public int id { get; set; }
    /// <summary>
    /// 分组名
    /// </summary>
    public string name { get; set; }
    /// <summary>
    /// 分组内用户数量
    /// </summary>
    public int count { get; set; }
}
```

在上述代码中，count 属性是为了兼容查询分组接口添加的，在映射创建分组接口时可忽略。



然后在项目中添加文件夹 `UserManager`，和用户管理相关的类与方法都封装在这个文件夹中。在 `UserManager` 文件夹中创建类 `UserGroup`，用于封装用户分组相关的操作。在类 `UserGroup` 中添加方法 `Create`，功能是创建一个分组，如代码清单 4-3 所示。

代码清单 4-3

```
public static UserGroupEntity Create(string name, string accessToken)
{
    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/create?access_
token={0}", accessToken);
    var obj = new { group = new { name = name } };
    var Jobj = Utils.PostResult<JObject>(obj, url);
    JToken errcode = null;
    errcode = Jobj.GetValue("errcode");
    var ret = new UserGroupEntity();
    if (errcode == null) //判断是否存在错误返回码。如果不存在，则说明分组创建成功
    {
        ret.ErrCode = 0;
        ret.id = Jobj["group"]["id"].Value<int>();
        ret.name = Jobj["group"]["name"].Value<string>();
    }
    else
    {
        ret.ErrCode = errcode.Value<int>();
    }
    return ret;
}
```

4.1.2 查询所有分组

查询所有分组的接口地址如下：

`https://api.weixin.qq.com/cgi-bin/groups/get?access_token=ACCESS_TOKEN`

HTTP 请求方式：GET。

在正确请求后，返回的 JSON 数据包如代码清单 4-4 所示。



代码清单 4-4

```
{
  "groups": [
    {
      "id": 0,
      "name": "未分组",
      "count": 3
    },
    {
      "id": 1,
      "name": "黑名单",
      "count": 0
    }
  ]
}
```

根据上述数据包创建映射实体，如代码清单 4-5 所示。

代码清单 4-5

```
public class UserGroups:ErrorEntity
{
    public List<UserGroupEntity> groups { get; set; }
}
```

返回实体创建好后，只需发送 GET 请求到查询所有分组的接口，然后将返回的数据包解析为类型是 UserGroups 的对象即可，如代码清单 4-6 所示。

代码清单 4-6

```
public static UserGroups QueryAll(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/groups/get"
            "?access_token={0}", accessToken);
    return Utils.GetResult<UserGroups>(url);
}
```




4.1.3 查询用户所在的分组

用户分组管理，归根到底还是对用户的管理，所以查询用户所在的分组也是一个比较基础且重要的接口了。接口地址如下：

`https://api.weixin.qq.com/cgi-bin/groups/getid?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

查询用户所在的分组，关键词在“用户”，而在公众号里标识用户唯一性的字段是 `openid`，所以请求该接口需要 `openid`，需要 POST 的数据格式如下：

```
{"openid": "od8XIjsmk6QdVTETa9jLtGWA6KBc"}
```

在请求成功后，正常情况下返回的 JSON 数据包格式如下：

```
{ "groupid": 102 }
```

为了统一接口调用，以及在消息调用出错时可正确返回错误信息，创建 `QueryGroupEntity`，并继承 `ErrorEntity`，如代码清单 4-7 所示。

代码清单 4-7

```
/// <summary>
/// 根据 openid 查询分组 ID 返回的实体
/// </summary>
public class QueryGroupEntity:ErrorEntity
{
    public int groupid { get; set; }
}
```

查询用户所在分组的代码如代码清单 4-8 所示。

代码清单 4-8

```
public static QueryGroupEntity QueryByOpenID(string OpenID, string
accessToken)
{
    var url =
```



```
string.Format("https://api.weixin.qq.com/cgi-bin/groups/getid?access_token={0}", accessToken);  
return Utils.PostResult<QueryGroupEntity>(new { openid = OpenID }, url);  
}
```

4.1.4 修改分组名

修改用户分组的接口地址如下所示：

https://api.weixin.qq.com/cgi-bin/groups/update?access_token=ACCESS_TOKEN

HTTP 请求方式： POST。

需要请求的参数包括需要修改的分组 ID 以及修改后的分组名。需要注意的是，分组名的长度为 30 个字符以内。POST 格式如下：

```
{"group":{"id":108,"name":"test2_modify2"}}
```

具体的实现代码如代码清单 4-9 所示。

代码清单 4-9

```
public static ErrorEntity Update(int id, string name, string accessToken)  
{  
    var url =  
string.Format("https://api.weixin.qq.com/cgi-bin/groups/update?access_token={0}", accessToken);  
    var obj = new  
    {  
        group = new { id = id, name = name }  
    };  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

4.1.5 移动用户分组

移动用户到指定分组的接口地址如下：

https://api.weixin.qq.com/cgi-bin/groups/members/update?access_token=ACCESS_TOKEN



HTTP 请求方式: POST。

需要传入的数据包括用户的 openid 以及目标分组 id。

POST 数据格式如下所示:

```
{"openid": "oDF3iYx0ro3_7jD4HFRDfrjdCM58", "to_groupid": 108}
```

具体的实现代码如代码清单 4-10 所示。

代码清单 4-10

```
public static ErrorEntity UpdateOpenIdToGroup(string OpenID, int to_groupid,
string accessToken)
{
    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/members/update?
access_token={0}", accessToken);
    var obj = new
    {
        openid = OpenID,
        to_groupid = to_groupid
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

而有的时候会有批量移动用户到指定分组的场景, 批量移动用户的接口地址如下:

https://api.weixin.qq.com/cgi-bin/groups/members/batchupdate?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

传入需要移动的用户 openid 列表, 以及目标分组 id。格式如下:

```
{"openid_list": ["oDF3iYx0ro3_7jD4HFRDfrjdCM58",
"oDF3iY9FGSSSRHom3B-0w5j4jlEyY"], "to_groupid": 108}
```

具体的实现代码如代码清单 4-11 所示。



代码清单 4-11

```
public static ErrorEntity UpdateOpenIdListToGroup(List<string> openid_list,
int to_groupid, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/groups/members/
batchupdate?access_token={0}", accessToken);
    var obj = new
    {
        openid_list = openid_list,
        to_groupid = to_groupid
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

4.2 用户信息管理

在关注者与公众号产生消息交互后，公众号可获得关注者的 **openid**（加密后的微信号，每个用户对每个公众号的 **openid** 是唯一的。对于不同公众号，同一用户的 **openid** 不同）。公众号可通过本接口来根据 **openid** 获取用户基本信息，包括昵称、头像、性别、所在城市、语言和关注时间等。如果开发者有多个公众号，或在公众号、移动应用之间统一用户账号的需求，则需要前往微信开放平台绑定公众号后，才可以利用 **UnionID** 机制满足上述需求。另外，只有认证的公众号才能绑定在微信开放平台。

UnionID 机制说明：开发者可通过 **openid** 来获取用户基本信息。在获取用户基本信息时，如果当前公众号绑定了微信开发平台，那么在绑定的开放平台账号下的移动应用、网站应用和公众账号，用户的 **UnionID** 是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，**UnionID** 是相同的。

4.2.1 获取用户基本信息

开发者可通过用户 **openid** 来获取用户基本信息。获取用户基本信息的接口地址如下：

https://api.weixin.qq.com/cgi-bin/user/info?access_token=ACCESS_TOKEN&openid=openid



HTTP 请求方式: GET。

正常情况下, 请求成功后, 返回的 JSON 数据包如代码清单 4-12 所示。

代码清单 4-12

```
{
  "subscribe": 1,
  "openid": "o6_bmjrrPTlm6_2sgVt7hMZOPfL2M",
  "nickname": "Band",
  "sex": 1,
  "language": "zh_CN",
  "city": "广州",
  "province": "广东",
  "country": "中国",
  "headimgurl":
"http://wx.qlogo.cn/mmopen/g3MonUZtNHkdmzicIlibx6iaFqAc56vxsLSUf6n5WKSXYVY0C
hQKkiaJSgQldZuTOgvLLrhJbERQQ4eMsv84eavHiaiceqxiBjXcFHe/0",
  "subscribe_time": 1382694957,
  "unionid": " o6_bmasdasdsad6_2sgVt7hMZOPfL",
  "remark": ""
}
```

根据 JSON 数据包创建用户信息实体, 如代码清单 4-13 所示。

代码清单 4-13

```
public class UserInfo:ErrorEntity
{
  /// <summary>
  /// 是否关注
  /// </summary>
  public int subscribe { get; set; }
  public string openid { get; set; }
  /// <summary>
  /// 昵称
  /// </summary>
  public string nickname { get; set; }
  /// <summary>
  /// 性别
  /// </summary>
  public int sex { get; set; }
  /// <summary>
```



```
    /// 语言
    /// </summary>
    public string language { get; set; }
    /// <summary>
    /// 城市
    /// </summary>
    public string city { get; set; }
    /// <summary>
    /// 广东
    /// </summary>
    public string province { get; set; }
    /// <summary>
    /// 中国
    /// </summary>
    public string country { get; set; }
    /// <summary>
    /// 图像
    /// </summary>
    public string headimgurl { get; set; }
    /// <summary>
    /// 关注时间
    /// </summary>
    public int subscribe_time { get; set; }
    public string unionid { get; set; }
    /// <summary>
    /// 用户备注
    /// </summary>
    public string remark { get; set; }
}
```

实体创建完毕后，在 `UserManager` 文件夹中创建类 `BaseUser`，用于封装操作微信用户信息的方法。将获取用户基本信息的代码添加到 `BaseUser` 类中，如代码清单 3-14 所示。

代码清单 4-14

```
public static UserInfo GetUserInfo(string openid, string access_token)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/user/info?access_
        token={0}&openid={1}",
        access_token, openid);
    return Utils.GetResult<UserInfo>(url);
}
```



4.2.2 设置用户备注名

开发者可以通过该接口对指定用户设置备注名,该接口暂时开放给微信认证的服务号。接口地址如下:

`https://api.weixin.qq.com/cgi-bin/user/info/updateremark?access_token=ACCESS_TOKEN`

HTTP 请求方式: POST。

请求示例: `{"openid":"oDF3iY9ffA-hqb2vVvbr7qxf6A0Q","remark":"pangzi"}`。

具体实现如代码清单 4-15 所示。

代码清单 4-15

```
public static ErrorEntity UpdateRemark(string openid, string remark, string
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/user/info/
updateremark?access_token={0}", accessToken);
    var obj = new
    {
        openid = openid, remark = remark
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

4.2.3 获取用户列表

公众号可通过获取用户列表接口来获取账号的关注者列表,关注者列表由一串 openid 组成。一次拉取调用最多拉取 10 000 个关注者的 openid,可以通过多次拉取的方式来满足组需求。接口的调用地址如下:

`https://api.weixin.qq.com/cgi-bin/user/get?access_token=ACCESS_TOKEN&next_openid=`
`NEXT_OPENID`

HTTP 请求方式: GET。



其中, 参数 `next_openid` 表示的是第一个拉取的 `openid`。不填的话则默认从头开始拉取。

正确时返回的 JSON 数据包如下:

```
{ "total": 2, "count": 2, "data": { "openid": [ "", "openid1", "openid2" ] }, "next_openid": "NEXT_OPENID" }
```

根据返回的数据包创建实体映射, 如代码清单 4-16 所示。

代码清单 4-16

```
public class UserListEntity : ErrorEntity
{
    /// <summary>
    /// 关注该公众账号的总用户数
    /// </summary>
    public int total { get; set; }
    /// <summary>
    /// 拉取的 openid 个数, 最大值为 10 000
    /// </summary>
    public int count { get; set; }
    /// <summary>
    /// 列表数据, openid 的列表
    /// </summary>
    public OpenidList data { get; set; }
    /// <summary>
    /// 拉取列表最后一个用户的 openid
    /// </summary>
    public string next_openid { get; set; }
}

public class OpenidList
{
    public List<string> openid { get; set; }
}
```

在调用接口时, 当公众号关注者数量超过 10 000 时, 可通过填写 `next_openid` 的值, 从而以多次拉取列表的方式满足需求。具体而言, 就是在调用接口时, 将上一次调用得到的返回中的 `next_openid` 值, 作为下一次调用中的 `next_openid` 值。在代码的实现过程中, 可以使用递归的方式。首先, 在第一次调用之后, 判断关注者的数量和此次获取到的用户



数量。如果关注者数量大于 10 000，而此次获取的数量正好是 10 000，则说明可能还有未获取的，此时则递归调用，然后将再次获取到的关注者列表和上次调用获取到的进行合并。具体实现如代码清单 4-17 所示。

代码清单 4-17

```
public static UserListEntity GetUserList(string accessToken, string
next_openid="")
{
    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/user/
get?access_token={0}&next_openid={1}", accessToken, next_openid);
    var retdata = Utils.GetResult<UserListEntity>(url);
    //判断调用是否成功。当调用成功，且总关注人数大于 10 000，本次获取到的用户数量等于
//10 000 时，则说明有尚未获取到的用户，此时递归调用，添加到列表
    if (retdata.ErrCode == 0 && retdata.total > 10000 && retdata.count ==
10000)
    {
        retdata.data.openid.AddRange(GetUserList(accessToken,
retdata.next_openid).data.openid);
    }
    return retdata;
}
```

4.3 OAuth 网页授权获取用户基本信息

OAuth 协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是 OAuth 的授权不会使第三方触及到用户的账户信息（如用户名与密码），即第三方无须使用用户的用户名与密码就可以申请获得该用户资源的授权，因此 OAuth 是安全的。如果用户在微信客户端中访问第三方网页，则公众号可以通过微信网页授权机制，来获取用户基本信息，进而实现业务逻辑。

4.3.1 网页授权回调域名设置

用户在网页授权页同意授权给公众号后，微信会将授权数据传给一个回调页面，回调页面所在的域名需要在微信公众平台管理后台进行登记，以确保安全可靠。所以在微信公众平台



求用户网页授权之前，开发者需要先到公众平台官网中的“开发者中心”页面配置授权回调域名，如图 4-1 所示。

微信小店	微信小店接口	-	已获得	
微信卡包	微信卡包接口	-	已获得	
设备功能	设备功能接口		未获得 ?	申请
网页账号	网页授权获取用户基本信息	无上限	已获得	修改

图 4-1

单击“修改”按钮后，弹出“OAuth2.0 网页授权”对话框，如图 4-2 所示。

OAuth2.0网页授权

授权回调页面域名:

ypyle.xicp.net

用户在网页授权页同意授权给公众号后，微信会将授权数据传给一个回调页面，回调页面需在此域名下，以确保安全可靠。沙盒号回调地址支持域名和ip，正式公众号回调地址只支持域名。

确认

取消

图 4-2

需要注意的是，好多人总是弄不清域名和 URL 的区别。比如说，<http://www.baidu.com/> 就是一个 URL，而 www.baidu.com 才是一个域名。授权回调域名配置规范为全域名，也就是配置了回调域名后，那么在此域名下的所有页面都可以进行 OAuth2.0 授权。比如配置的授权域名是 www.abc.com，那么 <http://www.abc.com/123.html>、<http://www.abc.com/345.html> 都是可以授权；而 http://123.abc.com、http://345.abc.com 由于不在 www.abc.com 的



域名下，因此无法进行 OAuth2.0 授权。

公众号的网页授权有两种处理方式。一种是以 `snsapi_base` 为 `scope` 发起的网页授权，是用来获取进入页面的用户的 `openid` 的，在获取到用户的 `openid` 后，调用获取用户基本信息接口，获取用户的详细信息。此种方式是静默授权并且自动跳转到回调页的。给用户的感觉就是直接进入了想进的页面；但弊端就是，如果用户并没有关注公众号，则是无法使用“获取用户基本信息”接口来获取用户信息的。另一种是以 `snsapi_userinfo` 为 `scope` 发起的网页授权，是用来获取用户的基本信息的。但这种授权在当用户单击授权链接后，会跳转到一个询问用户是否同意授权的页面，如图 4-3 所示。

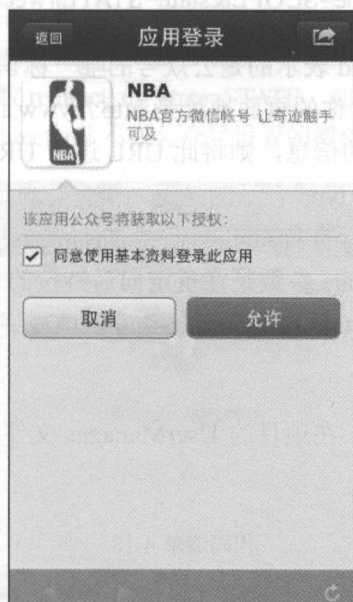


图 4-3

由于这种方式用户已经同意过了，所以无须关注，就可在授权后获取该用户的基本信息。并且，对于已关注公众号的用户，如果用户从公众号的会话或者自定义菜单进入本公众号的网页授权页，即使是 `scope` 为 `snsapi_userinfo`，也是静默授权，用户无感知。

具体而言，网页授权的流程分为四步：

- (1) 引导用户进入授权页面同意授权，获取 `code`。
- (2) 通过 `code` 换取网页授权 `access_token`（与基础支持中的 `access_token` 不同）。



(3) 如果需要, 开发者可以刷新网页授权 `access_token`, 避免过期。

(4) 通过网页授权 `access_token` 和 `openid` 获取用户基本信息 (支持 UnionID 机制)。

4.3.2 同意授权, 获取 code

在确保微信公众账号拥有授权作用域的权限的前提下 (服务号获得高级接口后, 默认拥有此权限), 填写授权域名后, 引导用户打开如下页面:

`https://open.weixin.qq.com/connect/oauth2/authorize?appid=APPID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE#wechat_redirect`

在上面的链接参数中, `appid` 表示的是公众号的唯一标识。`redirect_uri` 表示的是授权后重定向的回调链接地址。比如, 你的页面地址是 `http://www.abc.com/getuser.aspx`, 你想在微信用户进入此页面时获取用户的信息, 则将此 URL 进行 URL 编码后, 作为上述授权页面 `redirect_uri` 参数的值。`response_type` 表示的是返回类型, 固定为 `code`。`scope` 表示的是应用授权作用域, 可选值为上文介绍的 `snsapi_base` 或 `snsapi_userinfo`。至于如何选择, 开发者可根据实际使用场景来抉择。`state` 参数是在重定向后链接的 `state` 参数的值。在授权重定向后, 会在页面地址的后面带上 `state` 参数。开发者可以填写字母、数字或字母数字的组合作为参数值, 最多为 128 字节。

为了方便获取到授权链接, 在项目的 `UserManager` 文件夹中创建类 `OAuth`, 在类中添加如代码清单 4-18 所示的代码。

代码清单 4-18

```
public static string GetAuthUrl(string appid, string redirect_uri, string
state, AuthType authType = AuthType.snsapi_base)
{
    var url =
        "https://open.weixin.qq.com/connect/oauth2/authorize?appid={0}
&redirect_uri={1}&response_type=code&scope={2}&state={3}#wechat_redirect";
    return string.Format(url, appid, Utils.UrlEncode(redirect_uri),
authType, state);
}
```

在上述代码中, 参数 `AuthType` 是一个枚举类型, 有两个项, 分别是 `snsapi_base` 和 `snsapi_userinfo`。默认为 `AuthType.snsapi_base`, 即 `scope` 的值默认为 `snsapi_base`。`Utils.UrlEncode(redirect_uri)` 的功能是将字符串进行 URL 编码, 如代码清单 4-19 所示。



代码清单 4-19

```
public static string UrlEncode(string str)
{
    if (string.IsNullOrEmpty(str))
    {
        return "";
    }
    str = str.Replace("'", "");
    return HttpContext.Current.Server.UrlEncode(str);
}
```

在正确设置授权域名后，当用户访问授权链接时，页面将跳转到 `redirect_uri/?code=CODE&state=STATE`。在 `snsapi_userinfo` 模式下，若用户禁止授权，则重定向后不会带上 `code` 参数，仅会带上 `state` 参数 `redirect_uri?state=STATE`。如果用户访问授权链接时，出现了如图 4-4 所示的页面，则说明 `redirect_uri` 不在公众号设置的授权域名下，或未设置授权域名。

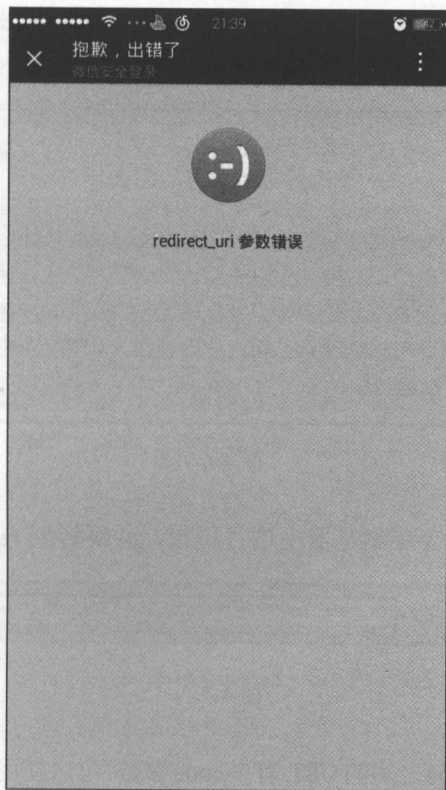


图 4-4



授权链接只能在微信客户端浏览时才有效。如果在非微信客户端浏览的话,则会出现如图 4-5 所示的提示。

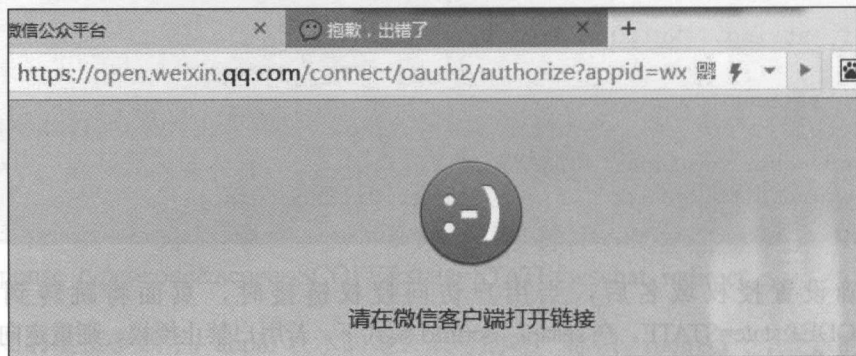


图 4-5

在这里有个小技巧,如果将授权链接从微信 PC 客户端或微信网页版中打开时,则可正常跳转。图 4-6 所示的是将授权链接从微信 PC 客户端发送给“文件传输助手”。

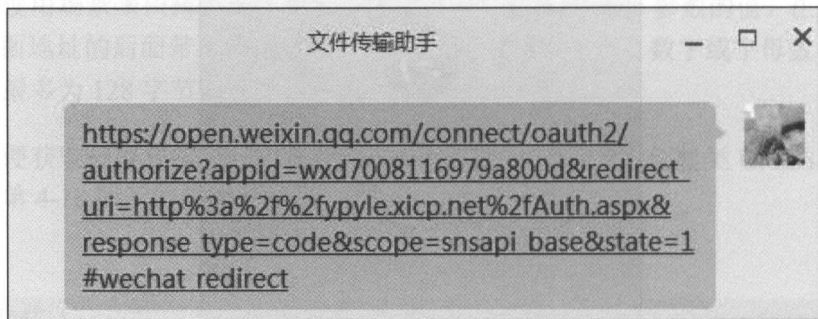


图 4-6

然后打开这个链接,将会在浏览器中进行授权,并跳转到 redirect_uri,如图 4-7 所示。

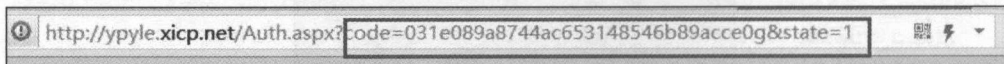


图 4-7

从图 4-7 中可看出,重定向之后的 URL 有个 code 参数,可以使用 `Request.QueryString["code"]` 获取 code 的值。code 作为换取 access_token 的票据,每次用户授权带上的 code 将不一样,code



只能使用一次, 5 分钟未被使用则自动过期。

4.3.3 通过 code 换取网页授权 access_token

请注意, 这里所说的 access_token 与基础支持中的 access_token 不同。获取网页授权 access_token 的接口地址如下:

```
https://api.weixin.qq.com/sns/oauth2/access_token?appid=APPID&secret=SECRET&code=CODE&grant_type=authorization_code
```

分别传入 appid、secret、code 进行 HTTP 的 GET 请求。正常情况下将返回代码清单 4-20 所示的 JSON 数据包。

代码清单 4-20

```
{
  "access_token": "ACCESS_TOKEN",
  "expires_in": 7200,
  "refresh_token": "REFRESH_TOKEN",
  "openid": "openid",
  "scope": "SCOPE",
  "unionid": "o6_bmasdasdsad6_2sgVt7hMZOPfL"
}
```

根据 JSON 数据包创建实体类 OAuthToken, 如代码清单 4-21 所示。

代码清单 4-21

```
public class OAuthToken : ErrorEntity
{
    /// <summary>
    /// 网页授权接口调用凭证。注意: 此 access_token 与基础支持的 access_token 不同
    /// </summary>
    public string access_token { get; set; }
    private int _expires_in;
    /// <summary>
    /// access_token 接口调用凭证超时时间, 单位 (秒)
    /// </summary>
    public int expires_in
    {

```



```
get { return _expires_in; }
set
{
    expires_time = DateTime.Now.AddSeconds(value);
    _expires_in = value;
}
}

/// <summary>
/// 用户刷新 access_token
/// </summary>
public string refresh_token { get; set; }
/// <summary>
/// 用户唯一标识。请注意，在未关注公众号时，用户访问公众号的网页，也会产生一个用户
/// 和公众号唯一的 openid
/// </summary>
public string openid { get; set; }
/// <summary>
/// 用户授权的作用域，使用逗号（,）分隔
/// </summary>
public string scope { get; set; }

public DateTime expires_time { get; set; }
/// <summary>
/// 只有在用户将公众号绑定到微信开放平台账号后，才会出现该字段
/// </summary>
public string unionid { get; set; }
}
```

从上述代码清单中可以看到，在获取网页授权 `access_token` 的同时，也获取到了 `openid`。如果网页授权的作用域为 `snsapi_base`，则可使用 `openid`，调用“获取用户基本信息”接口获取用户信息，`snsapi_base` 式的网页授权流程即到此为止。

另外，公众平台提供了检验授权凭证（`access_token`）是否有效的接口，在需要验证授权凭证有效的场景中可调用此接口。接口地址如下：

https://api.weixin.qq.com/sns/auth?access_token=access_token&openid=openid

HTTP 请求方式：GET。



正确的 JSON 返回结果是: { "errcode":0,"errmsg":"ok"}。

实现代码如代码清单 4-22 所示。

代码清单 4-22

```
public static ErrorEntity CheckAuthToken(string authtoken, string openid)
{
    var url =
        string.Format("https://api.weixin.qq.com/sns/auth?access_token={0}&openid={1}", authtoken, openid);
    return Utils.GetResult<ErrorEntity>(url);
}
```

4.3.4 刷新 access_token

由于 access_token 拥有较短的有效期 (7200 秒), 所以当 access_token 超时后, 可以使用 refresh_token 进行刷新。refresh_token 拥有较长的有效期 (7 天、30 天、60 天、90 天)。当 refresh_token 失效后, 需要用户重新授权。接口地址如下:

https://api.weixin.qq.com/sns/oauth2/refresh_token?appid=APPID&grant_type=refresh_token&refresh_token=REFRESH_TOKEN

HTTP 请求方式: GET。

接口地址中的 refresh_token 指的是在通过 code 获取授权凭证 access_token 时返回的 refresh_token 参数。在调用成功后, 返回的信息也和获取授权凭证的接口一致 (由于网页授权目前没有调用次数的限制, 所以刷新 access_token 的接口目前来讲使用的场景并不多)。

具体实现如代码清单 4-23 所示。

代码清单 4-23

```
public static OAuthToken GetRefreshToken(string appid,
    string refresh_token)
{
    var url =
        string.Format("https://api.weixin.qq.com/sns/oauth2/refresh_token?appid={0}&grant_type=refresh_token&refresh_token={1}", appid, refresh_token);
}
```



```
return Utils.GetResult<OAuthToken>(url);  
}
```

4.3.5 拉取用户信息

如果网页授权作用域为 `snsapi_userinfo`，则此时开发者可以通过 `access_token` 和 `openid` 拉取用户信息了。接口地址如下：

`https://api.weixin.qq.com/sns/userinfo?access_token=ACCESS_TOKEN&openid=openid&lang=zh_CN`

HTTP 请求方式：GET。

此接口正确时返回的 JSON 数据包和获取用户基本信息接口基本一致；不同的是此接口的返回信息多了个 `privilege` 属性，表示的是用户特权信息，类型是字符串数组，如微信沃卡用户为 (`chinaunicom`)。所以只需在 4.1.1 节中讲到的类 `UserInfo` 中添加属性 `privilege` 即可。代码清单 4-24 为获取用户信息的方法。

代码清单 4-24

```
public static UserInfo GetUserInfo(string authtoken, string openid)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/sns/userinfo?access_token=  
{0}&openid={1}&lang=zh_CN", authtoken, openid);  
    return Utils.GetResult<UserInfo>(url);  
}
```

请注意：使用此接口是无法获取用户是否关注及关注时间的。也就是说，类 `UserInfo` 中的 `subscribe` 和 `subscribe_time` 属性在此接口中是无效的。

第5章 多客服接口

5.1 多客服简介与开通

多客服为公众号提供客服功能，支持多人同时为一个公众号提供客服服务，支持会话自动接入。当有新客户发送消息时，多客服优先分配给上次接入的工号。如果客户第一次接入或者上次接入的工号不在线，则在未达到接待上限的在线工号中轮流平均分配。还支持转接会话。当客户有问题需要其他客服回答时，可以选择转接会话，转接当前客户至其他在线客服处接受咨询。通过微信认证的公众号在公众平台后台依次进入“功能”→“添加功能插件”→“多客服”，进入多客服页面，如图 5-1 和图 5-2 所示。

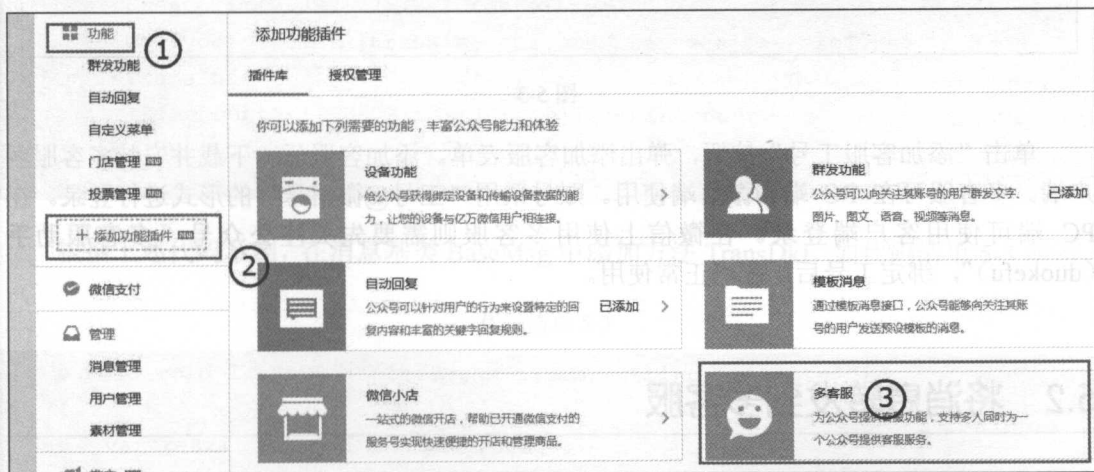


图 5-1

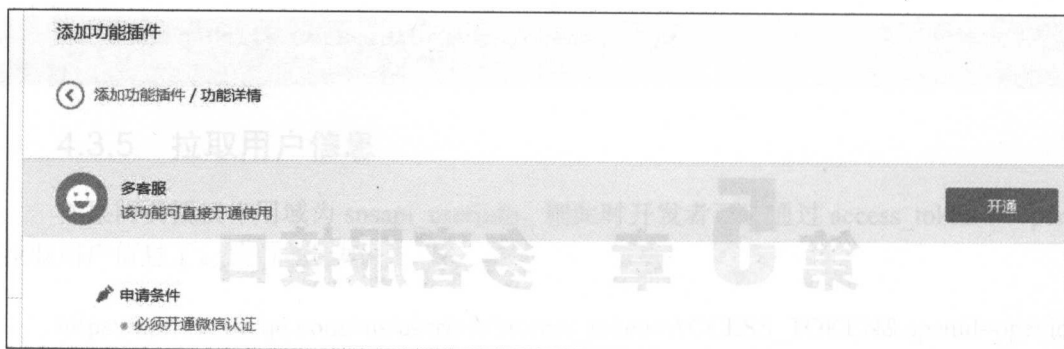


图 5-2

多客服开通后，在公众平台后台“功能”栏中会出现“多客服”链接，单击该链接可进入多客服下载与配置页面，如图 5-3 所示。

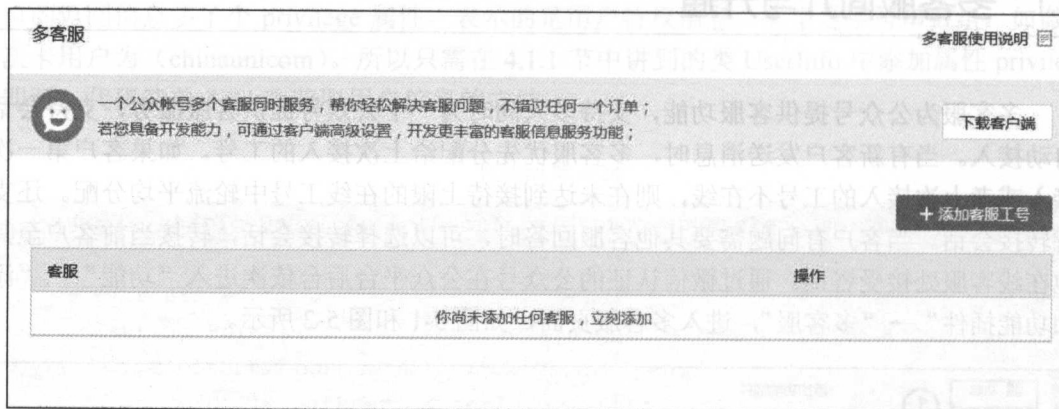


图 5-3

单击“添加客服工号”按钮，弹出添加客服表单。添加客服后，下载并安装多客服客户端。多客服可在 PC 端和微信端使用。账号采用“工号@微信号”的形式进行登录。在 PC 端可使用客户端登录。在微信上使用多客服则需要先关注公众号“多客服助手(duokefu)”，绑定工号后，才可正常使用。

5.2 将消息转发到多客服

如果公众号处于开发模式，普通微信用户向公众号发消息时，微信服务器会先将消息



POST 到开发者填写的 URL 上。如果希望将消息转发到多客服系统,则需要开发者在响应包中返回 `MsgType` 为 `transfer_customer_service` 的消息,微信服务器收到响应后会把当次发送的消息转发至多客服系统。消息被转发到多客服以后,会被自动分配给一个在线的客服账号,示例代码如代码清单 5-1 所示。你也可以在返回 `transfer_customer_service` 消息时,在 XML 中附上 `TransInfo` 信息指定分配给某个客服账号,如代码清单 5-2 所示。用户被客服接入以后,在客服关闭会话以前,处于会话过程中时,用户发送的消息均会被直接转发至客服系统。当会话超过 2 小时客服没有关闭时,微信服务器会自动停止转发至多客服,而将消息恢复发送至开发者填写的 URL 上。用户在等待队列中时,用户发送的消息仍然会被推送至开发者填写的 URL 上。

代码清单 5-1 (转发到多客服,随机分配客服)

```
<xml>
  <ToUserName><![CDATA[touser]]></ToUserName>
  <FromUserName><![CDATA[fromuser]]></FromUserName>
  <CreateTime>1399197672</CreateTime>
  <MsgType><![CDATA[transfer_customer_service]]></MsgType>
</xml>
```

代码清单 5-2 (转发消息到指定客服)

```
<xml>
  <ToUserName><![CDATA[touser]]></ToUserName>
  <FromUserName><![CDATA[fromuser]]></FromUserName>
  <CreateTime>1399197672</CreateTime>
  <MsgType><![CDATA[transfer_customer_service]]></MsgType>
  <TransInfo>
    <KfAccount><![CDATA[test1@test]]></KfAccount>
  </TransInfo>
</xml>
```

根据上述代码示例,在消息基类 `BaseMsg` 中添加方法 `TransDkf`,如代码清单 5-3 所示。

代码清单 5-3

```
public void TransDkf(EnterParam param, string account="")
{
    var resxml = new StringBuilder();
    resxml.AppendFormat("<xml><ToUserName><![CDATA[{0}]]></ToUserName>",
        FromUserName);
```



```
resxml.AppendFormat("<FromUserName><![CDATA[{0}]]></FromUserName>",  
ToUserName);  
resxml.AppendFormat("<CreateTime>{0}</CreateTime>",  
Utils.ConvertDateTimeInt(DateTime.Now));  
resxml.Append("<MsgType><![CDATA[transfer_customer_service]]></MsgType>");  
if (account != "")  
{  
    resxml.AppendFormat("<TransInfo><KfAccount><![CDATA[{0}]]>  
</KfAccount></TransInfo>", account);  
}  
resxml.AppendFormat("</xml>");  
Response(param, resxml.ToString());  
}
```

开发者只需要在需要的时候调用 `TransDkf` 方法。需要注意的是，多客服转发时如果指定的客服没有接入能力（不在线、没有开启自动接入或者自动接入已满），该用户会一直等待指定客服有介入能力后才会被接入，而不会被其他客服接待。建议在指定客服时先查询客服的接入能力，指定到有能力接入的客服，保证客户能够及时得到服务。

下面的示例演示的是调用 `TransDkf` 方法，将用户发送的所有文本消息转发到多客服进行处理。

在处理程序的页面 `wx.ashx` 的处理文本消息的方法中，添加如下代码：

```
baseMsg.TransDkf(param);
```

此处未指定客服，如图 5-4 所示。

```
/// <summary>  
/// 文本消息处理程序  
/// </summary>  
1 个引用  
private void TextHandler(BaseMsg baseMsg)  
{  
    baseMsg.TransDkf(param);  
}
```

图 5-4

此时用微信发消息给公众平台，多客服将会收到用户发送的消息，且可以直接进行回复，如图 5-5 和图 5-6 所示。

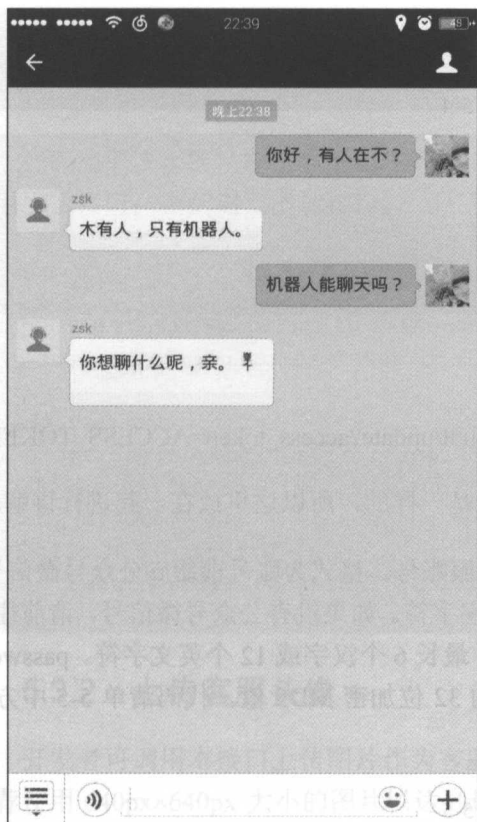


图 5-5

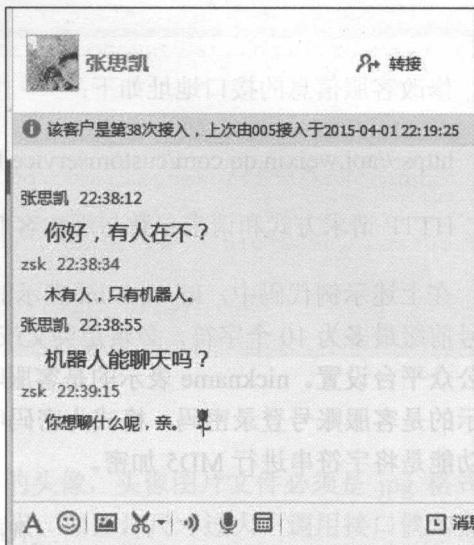


图 5-6

5.3 客服管理

5.3.1 设置客服账号

设置客服账号包括添加与修改，每个公众号最多添加 10 个客服账号。开发者可通过该接口为公众号添加客服账号。接口地址如下：

https://api.weixin.qq.com/customservice/kfaccount/add?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。



请求示例如代码清单 5-4 所示。

代码清单 5-4

```
{
    "kf_account" : test1@test,
    "nickname" : "客服 1",
    "password" : "pswmd5",
}
```

修改客服信息的接口地址如下：

https://api.weixin.qq.com/customservice/kfaccount/update?access_token=ACCESS_TOKEN

HTTP 请求方式和请求参数与添加客服接口是一样的，所以这里放在一起进行讲解。

在上述示例代码中，kf_account 表示的是客服账号，格式为账号前缀@公众号微信号，账号前缀最多为 10 个字符，必须是英文或者数字字符。如果没有公众号微信号，请前往微信公众平台设置。nickname 表示的是客服昵称，最长 6 个汉字或 12 个英文字符。password 表示的是客服账号登录密码，格式为密码明文的 32 位加密 MD5 值。代码清单 5-5 中方法的功能是将字符串进行 MD5 加密。

代码清单 5-5

```
/// <summary>
/// MD5 加密
/// </summary>
/// <param name="pwd">要加密的字符串</param>
/// <param name="encoding">字符编码方法。默认 utf-8</param>
/// <returns>加密后的密文</returns>
public static string MD5(string pwd, string encoding="utf-8")
{
    MD5 md5 = new MD5CryptoServiceProvider();
    byte[] data = Encoding.GetEncoding(encoding).GetBytes(pwd);
    byte[] md5data = md5.ComputeHash(data);
    md5.Clear();
    string str = "";
    for (int i = 0; i < md5data.Length; i++)
    {
        str += md5data[i].ToString("x").PadLeft(2, '0');
    }
}
```




```
}  
return str;  
}
```

代码清单 5-6 的功能是判断 isUpdate 的值。如果为 true，则说明是更新客服信息，则调用更新接口；否则调用添加接口。

代码清单 5-6

```
public static ErrorEntity SetAccount(string account, string nickName, string  
password, string accessToken, bool isUpdate=false)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/customservice/kfaccount/  
{0}?access_token={1}", isUpdate?"update":"add", accessToken);  
    var obj = new { kf_account = account, nickname = nickName, password =  
        Utils.MD5(password) };  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

5.3.2 上传客服头像

开发者可调用本接口上传图片作为客服人员的头像，头像图片文件必须是 jpg 格式，推荐使用 640px×640px 大小的图片以达到最佳效果。如图片尺寸过大，调用接口偶尔会返回 501 错误。接口地址如下：

http://api.weixin.qq.com/customservice/kfaccount/uploadheadimg?access_token=ACCESS_TOKEN&kf_account=KFACCOUNT

HTTP 请求方式：POST/FORM。

在请求时，以 FORM 表单的形式将图片上传到微信服务器，可参考素材管理相关的接口实现方式。具体实现如代码清单 5-7 所示。

代码清单 5-7

```
public static ErrorEntity UploadHeadImg(string filepath, string account,  
string accessToken)  
{  
    var url =  
        string.Format("http://api.weixin.qq.com/customservice/kfaccount/
```



```
uploadheading?access_token={0}&kf_account={1}",
accessToken, account);
var formlist = new List<FormEntity> { new FormEntity {
    IsFile = true, Name = "media", Value = filepath } };
return Utils.PostFormResult<ErrorEntity>(formlist, url);
}
```

5.3.3 删除客服账号

删除客服的接口地址如下：

https://api.weixin.qq.com/customservice/kfaccount/del?access_token=ACCESS_TOKEN&kf_account=KFACCOUNT

HTTP 请求方式：GET。

接口中的 `kf_account` 参数必须是完整客服账号，格式为：账号前缀@公众号微信号。

实现代码如代码清单 5-8 所示。

代码清单 5-8

```
public static ErrorEntity DeleteAccount(string account, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/customservice/kfaccount/
del?access_token={0}&kf_account={1}", accessToken, account);
    return Utils.GetResult<ErrorEntity>(url);
}
```

5.3.4 获取在线客服接待信息

开发者通过本接口可以获取当前在线客服的接待信息，包括工号、客服登录账号、客服在线状态（手机在线、PC 客户端在线、手机和 PC 客户端全都在线）、客服自动接入最大值、客服当前接待客户数。开发者利用本接口提供的信息，在需要转发用户消息到多客服时，可以调用本接口来获取最适合接入的客服。结合会话记录，可以开发“在线客服实时服务监控”等功能。接口地址如下：

https://api.weixin.qq.com/cgi-bin/customservice/getonlinekflist?access_token=ACCESS_TOKEN



HTTP 请求方式: GET。

正常情况下, 请求接口后, 返回的 JSON 示例如代码清单 5-9 所示。

代码清单 5-9

```
{
  "kf_online_list": [
    {
      "kf_account": "test1@test",
      "status": 1,
      "kf_id": "1001",
      "auto_accept": 0,
      "accepted_case": 1
    }
  ]
}
```

根据上述代码, 在 ReceiveEntity 文件夹内新建实体类, 如代码清单 5-10 所示。

代码清单 5-10

```
using System.Collections.Generic;
namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// 多客服在线列表
    /// </summary>
    public class KfOnLineList : ErrorEntity
    {
        public List<OnLineInfo> kf_online_list { get; set; }
    }

    public class OnLineInfo
    {
        /// <summary>
        /// 完整客服账号, 格式为: 账号前缀@公众号微信号
        /// </summary>
        public string kf_account { get; set; }
        /// <summary>
```



```
/// 客服在线状态 1: PC 在线, 2: 手机在线。若 PC 和手机同时在线则为 1+2=3
/// </summary>
public string status { get; set; }
/// <summary>
/// 客服工号
/// </summary>
public string kf_id { get; set; }
/// <summary>
/// 客服设置的最大自动接入数
/// </summary>
public string auto_accept { get; set; }
/// <summary>
/// 客服当前正在接待的会话数
/// </summary>
public string accepted_case { get; set; }
}
}
```

返回实体创建完毕后, 将获取在线客服接待信息的方法添加到 CustomerServices 类中, 如代码清单 5-11 所示。

代码清单 5-11

```
public static KfOnLineList GetKfOnLineList(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/customservice/
getonlinekflist?access_token={0}", accessToken);
    return Utils.GetResult<KfOnLineList>(accessToken);
}
```

5.3.5 获取客服基本信息

开发者可通过本接口获取公众号所设置的客服基本信息, 包括客服工号、客服昵称、客服登录账号。开发者利用客服基本信息, 结合客服接待情况, 可以开发例如“指定客服接待”等功能。接口地址如下:

https://api.weixin.qq.com/cgi-bin/customservice/getkflist?access_token=ACCESS_TOKEN



HTTP 请求方式: GET。

请求成功后, 返回的实例如代码清单 5-12 所示。

代码清单 5-12

```
{
  "kf_list": [
    {
      "kf_account": "test1@test",
      "kf_headimgurl": "http://mmbiz.qpic.cn/mmbiz/4whpV1",
      "kf_id": "1001",
      "kf_nick": "ntest1"
    }
  ]
}
```

从上述代码清单中, 可以看出, kf_list 是个列表。根据上述代码创建实体类, 如代码清单 5-13 所示。

代码清单 5-13

```
public class KfList:ErrorEntity
{
    public List<KfInfo> kf_list { get; set; }
}
public class KfInfo
{
    /// <summary>
    /// 完整客服账号, 格式为: 账号前缀@公众号微信号
    /// </summary>
    public string kf_account { get; set; }
    /// <summary>
    /// 客服头像
    /// </summary>
    public string kf_headimgurl { get; set; }
    /// <summary>
    /// 客服工号
    /// </summary>
    public string kf_id { get; set; }
```



```
/// <summary>
/// 客服昵称
/// </summary>
public string kf_nick { get; set; }
}
```

具体实现代码如代码清单 5-14 所示。

代码清单 5-14

```
public static KfList GetKfList(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/customservice/
getkflist?access_token={0}", accessToken);
    return Utils.GetResult<KfList>(url);
}
```

5.3.6 获取客服聊天记录接口

在需要时,开发者可以通过获取客服聊天记录接口,获取多客服的会话记录,包括客服和用户会话的所有消息记录和会话的创建、关闭等操作记录。利用此接口可以开发如消息记录、工作监控、客服绩效考核等功能。

获取客服聊天记录的接口地址如下:

https://api.weixin.qq.com/cgi-bin/customservice/getrecord?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例如代码清单 5-15 所示。

代码清单 5-15

```
{
    "endtime" : 987654321,
    "openid" : "openid",
    "pageindex" : 1,
    "pagesize" : 10,
    "starttime" : 123456789
}
```



在上述代码清单中, openid 表示的是要查询的用户的 id, 可为空。如果为空, 则查询的是所有用户的聊天信息。endtime、starttime 分别表示的是查询的开始时间和结束时间, 格式为 UNIX 时间戳, 每次查询不能跨日查询。pagesize 表示的是每次拉取的数量, 每页最多拉取 1000 条。pageindex 表示的是查询第几页, 从 1 开始。

正常情况下, 调用接口后返回的 JSON 数据包如代码清单 5-16 所示。

代码清单 5-16

```
{
  "recordlist" : [
    {
      "openid" : "oDF3iY9WMaswOPWjCIp_f3Bnpljk",
      "opcode" : 2002,
      "text" : " 您好, 客服 test1 为您服务。",
      "time" : 1400563710,
      "worker" : "test1"
    },
    {
      "openid" : "oDF3iY9WMaswOPWjCIp_f3Bnpljk",
      "opcode" : 2003,
      "text" : "你好, 有什么事情?",
      "time" : 1400563731,
      "worker" : "test1"
    }
  ]
}
```

从代码中可以发现, time 属性也是 UNIX 格式的时间戳。为了方便调用, 在实体类中添加一个属性 Time。在给 time 属性赋值时, 需将 time 转换成 DateTime 格式后, 赋值给 Time。另外, opcode 表示的是操作 ID (会话状态), 对应信息如表 5-1 所示。

表 5-1

ID 值	说 明	ID 值	说 明
1000	创建未接入会话	1005	抢接会话
1001	接入会话	2001	公众号收到消息
1002	主动发起会话	2002	客服发送消息
1004	关闭会话	2003	客服收到消息



同理，为了更明确地展示出会话状态，在实体类中添加属性 `operstr`。在给 `opercode` 赋值时，根据表 5-1 的对应关系，同时给 `operstr` 赋值，如代码清单 5-17 所示。

代码清单 5-17

```
public class RecordList : ErrorEntity
{
    public List<RecordInfo> recordlist { get; set; }
}

public class RecordInfo
{
    /// <summary>
    /// 客服账号
    /// </summary>
    public string worker { get; set; }
    /// <summary>
    /// 用户的标识，对当前公众号唯一
    /// </summary>
    public string openid { get; set; }

    private string _opercode;

    /// <summary>
    /// 操作 ID（会话状态）
    /// </summary>
    public string opercode
    {
        get { return _opercode; }
        set
        {
            switch (value)
            {
                case "1000": operstr = "创建未接入会话"; break;
                case "1001": operstr = "接入会话"; break;
                case "1002": operstr = "主动发起会话"; break;
                case "1004": operstr = "关闭会话"; break;
                case "1005": operstr = "抢接会话"; break;
                case "2001": operstr = "公众号收到消息"; break;
                case "2002": operstr = "客服发送消息"; break;
            }
        }
    }
}
```




```
        case "2003": operstr = "客服收到消息"; break;

    }
    _opcode = value;
}

}

/// <summary>
/// 会话状态描述
/// </summary>
public string operstr { get; set; }

/// <summary>
/// 操作时间, UNIX 时间戳
/// </summary>
public int time
{
    get { return Utils.ConvertDateTimeInt(Time); }
    set { Time = Utils.UnixTimeToTime(value.ToString()); }
}

public DateTime Time { get; set; }
/// <summary>
/// 聊天记录
/// </summary>
public string text { get; set; }
}
```

上述代码中 `Utils.UnixTimeToTime` 方法的功能是将 UNIX 时间戳转换成 `DateTime`, 具体代码如下所示:

```
public static DateTime UnixTimeToTime(string timeStamp)
{
    DateTime dtStart = TimeZone.CurrentTimeZone.ToLocalTime(
        new DateTime(1970, 1, 1));
    long lTime = long.Parse(timeStamp + "0000000");
    TimeSpan toNow = new TimeSpan(lTime);
    return dtStart.Add(toNow);
}
```

获取客服聊天记录接口的实现代码如代码清单 5-18 所示。



代码清单 5-18

```
public static RecordList GetRecordList(DateTime startTime, DateTime endTime,
int pageIndex, int pageSize,
string accessToken, string openid = "")
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/customservice/
getrecord?access_token={0}", accessToken);
    var obj = new
    {
        endtime = Utils.ConvertDateTimeInt(endTime),
        starttime = Utils.ConvertDateTimeInt(startTime),
        openid = openid,
        pageIndex = pageIndex,
        pageSize = pageSize
    };
    return Utils.PostResult<RecordList>(obj, url);
}
```

5.4 多客服会话控制

5.4.1 会话状态通知事件

公众号开通多客服功能以后，当客服人员有接入会话、关闭会话、转接会话时，微信服务器会将会话对应的事件推送到公众号填写的 URL 上。下面的代码清单（代码清单 5-19 到代码清单 5-21）中是各会话事件推送的 XML 数据示例。

代码清单 5-19（接入会话）

```
<xml>
  <ToUserName><![CDATA[gh_218ed5d1b227]]></ToUserName>
  <FromUserName><![CDATA[oxb7QsiokOH17CwSwD37RDMSN978]]></FromUserName>
  <CreateTime>1428067918</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[kf_create_session]]></Event>
  <KfAccount><![CDATA[005@jkcy2014618]]></KfAccount>
</xml>
```



代码清单 5-20 (关闭会话)

```
<xml>
  <ToUserName><![CDATA[gh_218ed5d1b227]]></ToUserName>
  <FromUserName><![CDATA[oxb7QsiokOH17CwSWd37RDMSN978]]></FromUserName>
  <CreateTime>1428067918</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[kf_close_session]]></Event>
  <KfAccount><![CDATA[005@jkcy2014618]]></KfAccount>
</xml>
```

代码清单 5-21 (转接会话)

```
<xml>
  <ToUserName><![CDATA[gh_218ed5d1b227]]></ToUserName>
  <FromUserName><![CDATA[oxb7QsiokOH17CwSWd37RDMSN978]]></FromUserName>
  <CreateTime>1428067918</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[kf_switch_session]]></Event>
  <KfAccount><![CDATA[005@jkcy2014618]]></KfAccount>
</xml>
```

从上述数据包中可以发现,除了基础的事件类型的属性外,多客服会话状态通知还包含 KfAccount (客服工号) 属性。仿照 3.3 节 (接收消息) 中的相关处理方式,在项目 WxApi 中的 MsgEntity 文件夹添加实体类 KfEventMsg,如代码清单 5-22 所示。

代码清单 5-22

```
public class KfEventMsg:EventArgs
{
    public string KfAccount { get; set; }
}
```

然后在 MsgFactory 类中处理事件类型的 switch 语句中添加如图 5-7 所示的代码。

```
//自定义菜单扫描二维码
case EventType.SCANCODE_PUSH:
    case EventType.SCANCODE_WAITMSG:
        msg = Utils.ConvertObj<ScanMenuEventMsg>(postStr);break;
//客服会话控制
case EventType.KF_CLOSE_SESSION:
case EventType.KF_CREATE_SESSION:
case EventType.KF_SWITCH_SESSION:
    msg = Utils.ConvertObj<KfEventMsg>(postStr);break;
default:
    msg = Utils.ConvertObj<EventMsg>(postStr); break;
```

图 5-7



最后, 在 wx.ashx 页面中添加“绑定客服会话控制”的代码。由于 3 种客服会话事件的实体类型是一样的, 因此为了简化代码量, 3 种会话事件绑定了同一个处理方法, 在处理方法中再根据事件类型和具体的业务逻辑做相应的操作, 如图 5-8 所示。

```
#region 创建多客服会话消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new MsgHandlerEntity
{
    MessageType = MessageType.EVENT,
    EventType = EventType.KF_CREATE_SESSION,
    Action = KfHandler
});
#endregion
#region 关闭多客服会话消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new MsgHandlerEntity
{
    MessageType = MessageType.EVENT,
    EventType = EventType.KF_CLOSE_SESSION,
    Action = KfHandler
});
#endregion
#region 转接多客服会话消息处理绑定
MsgHandlerEntity.MsgHandlerEntities.Add(new MsgHandlerEntity
{
    MessageType = MessageType.EVENT,
    EventType = EventType.KF_SWITCH_SESSION,
    Action = KfHandler
});
});
```

图 5-8

其中, KfHandler 方法的代码实现如清单 5-23 所示。

代码清单 5-23

```
private void KfHandler(BaseMsg baseMsg)
{
    var msg = (KfEventMsg) baseMsg;
    //处理不同的会话事件
    switch (msg.Event)
    {
        case EventType.KF_CLOSE_SESSION:
            break;
        case EventType.KF_CREATE_SESSION:
            break;
    }
}
```




```
case EventType.KF_SWITCH_SESSION:
    break;
```

5.4.2 会话创建与关闭

开发者可以通过创建会话接口给未被接入的客户创建会话。由于此接口不会受客服自动接入数以及自动接入开关限制，所以可以直接将某个客户指定给客服工号接待。但只能为在线的客服（PC 客户端在线，或者已绑定多客服助手）创建会话。接口地址如下：

https://api.weixin.qq.com/customservice/kfsession/create?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例如代码清单 5-24 所示。

代码清单 5-24

```
{
    "kf_account" : "test1@test",
    "openid" : "openid",
    "text" : "这是一段附加信息"
}
```

在上述代码清单中，kf_account 表示的是客服工号，openid 指的是客户的 openid，text 则表示的是附加信息，当接入后，文本会展示在客服人员的多客服客户端。当调用成功后，返回的信息是：{"errcode":0,"errmsg":"ok"}。当 errcode 的值为 61458 时，表示客户正在被其他客服接待。为 61459 时，表示客服不在线。因为这两种错误码微信官方并没有添加到全局返回码中，所以为了方便调用，需要将这两种错误码信息添加到资源文件 CodeInfo 中，如图 5-9 所示。

CodeInfo.txt*	KfSessionStatus.cs	WaitCaseList.cs	KfOnLineList
61500	日期格式错误		
61501	日期范围错误		
61458	客户正在被其他客服接待 (customer accepted by xxx@xxxx)		
61459	客服不在线 (kf offline)		

图 5-9

同时，当需要关闭某个会话时，可以调用关闭会话接口，接口地址如下：



https://api.weixin.qq.com/customservice/kfsession/close?access_token=ACCESS_TOKEN

请求方式与创建会话的方式是一样的，在此不再赘述。从接口地址可以看出，创建会话和关闭会话的接口地址是相似的，在这里可以将创建和关闭会话封装在一个方法中，创建一个枚举 `KfSessionType`，封装两个选项，分别是 `create` 和 `close`，调用的时候传递一个枚举参数来表示请求的是创建会话，抑或是关闭会话，如代码清单 5-25 所示。

代码清单 5-25

```
/// <summary>
/// 客服会话创建与关闭
/// </summary>
/// <param name="kfAccount">客服工号</param>
/// <param name="openId">用户 ID</param>
/// <param name="text">附加文本</param>
/// <param name="sessionType">请求的会话状态</param>
public static ErrorEntity KfSession(string kfAccount, string openId, string
text, KfSessionType sessionType, string accessToken)
{
    var url = string.Format("https://api.weixin.qq.com/customservice/
kfsession/{0}?access_token={1}", sessionType.ToString(), accessToken);
    var obj = new { kf_account = kfAccount, openid = openId, text = text };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

调用时，当执行代码清单 5-26 所示的代码时，多客服客户端的效果如图 5-10 所示。

代码清单 5-26

```
CustomerServices.KfSession("003@jkcy2014618", "oxb7QsvLzXJTK6nBI9A8nz
38gWxM", "vip 客户，请认真对待。", KfSessionType.create, accessToken);
```

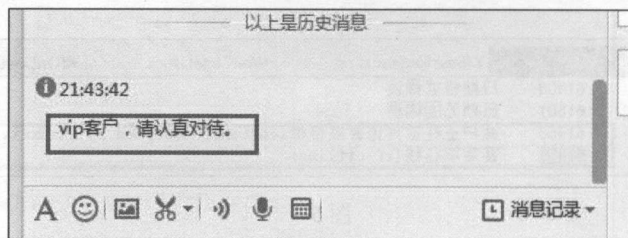


图 5-10



5.4.3 获取客户的会话状态

此接口的用途是获取某个客户是否接入到会话。接口地址如下：

`https://api.weixin.qq.com/customservice/kfsession/getsession?access_token=ACCESS_TOKEN&openid=openid`

HTTP 请求方式：GET。

调用成功后，返回示例如代码清单 5-27 所示。

代码清单 5-27

```
{
  "createtime" : 123456789,
  "errcode" : 0,
  "errmsg" : "ok",
  "kf_account" : "test1@test"
}
```

其中，kf_account 表示的是正在接待的客服，为空则表示没有人在接待。createtime 表示的是会话的接入时间，格式是 UNIX 时间戳。在调用时，需要将此参数转换成常规时间。与上述代码清单对应的实体类如代码清单 5-28 所示。

代码清单 5-28

```
public class KfSessionStatus : ErrorEntity
{
    /// <summary>
    /// 会话接入的时间，UNIX 时间戳
    /// </summary>
    public int createtime
    {
        get { return Utils.ConvertDateTimeInt(CreateTime); }
        set { CreateTime = Utils.UnixTimeToTime(value.ToString()); }
    }
    /// <summary>
    /// 正在接待的客服，为空则表示没有人在接待
    /// </summary>
}
```



```
public string kf_account { get; set; }  
/// <summary>  
/// 会话接入的时间  
/// </summary>  
public DateTime CreateTime { get; set; }  
}
```

代码清单 5-29 所示的是此接口的具体实现方法。

代码清单 5-29

```
public static KfSessionStatus GetsKfSessionStatus(string openId, string  
accessToken)  
{  
    var url = string.Format("https://api.weixin.qq.com/customservice/  
kfsession/getsession?access_token={0}&openid={1}", accessToken, openId);  
    return Utils.GetResult<KfSessionStatus>(url);  
}
```

5.4.4 获取客服的会话列表

获取客服的会话列表指的是指定客服当前正在服务的客户列表。接口地址如下：

https://api.weixin.qq.com/customservice/kfsession/getsessionlist?access_token=ACCESS_TOKEN&kf_account=KFACCOUNT

HTTP 请求方式：GET。

正确请求时返回的 JSON 数据包如代码清单 5-30 所示。

代码清单 5-30

```
{  
    "sessionlist" : [  
        {  
            "createtime" : 123456789,  
            "openid" : "openid"  
        },  
        {  
            "createtime" : 123456789,
```




```
"openid" : "openid"
    }
  ]
}
```

根据上述数据包创建实体类 KfSessionList，如代码清单 5-31 所示。

代码清单 5-31

```
public class KfSessionList:ErrorEntity
{
    public List<SessionInfo> sessionlist { get; set; }
}
public class SessionInfo
{
    /// <summary>
    /// 会话接入的时间，UNIX 时间戳
    /// </summary>
    public int createtime
    {
        get { return Utils.ConvertDateTimeInt(CreateTime); }
        set { CreateTime = Utils.UnixTimeToTime(value.ToString()); }
    }
    /// <summary>
    /// 会话接入的时间
    /// </summary>
    public DateTime CreateTime { get; set; }
    public string openid { get; set; }
}
```

创建实体完毕后，在 CustomerServices 类中添加方法 GetKfSessionList，返回类型为上述代码所示的实体类 KfSessionList，如代码清单 5-32 所示。

代码清单 5-32

```
public static KfSessionList GetKfSessionList(string kfAccount, string
accessToken)
{
    var url = string.Format("https://api.weixin.qq.com/customservice/
kfsession/getsessionslist?access_token={0}&kf_account={1}", accessToken,
```



```
kfAccount);  
    return Utils.GetResult<KfSessionList>(url);  
}
```

5.4.5 获取未接入会话列表

未接入会话列表指的是在等待队列中的会话列表,此接口最多返回最早进入队列的100个未接入会话。接口地址如下:

https://api.weixin.qq.com/customservice/kfsession/getwaitcase?access_token=ACCESS_TOKEN

HTTP 请求方式: GET。

正确请求后,返回的 JSON 字符串如代码清单 5-33 所示。

代码清单 5-33

```
{  
  "count" : 150,  
  "waitcaselist" : [  
    {  
      "createtime" : 123456789,  
      "kf_account" : "test1@test",  
      "openid" : "openid"  
    },  
    {  
      "createtime" : 123456789,  
      "kf_account" : "",  
      "openid" : "openid"  
    }  
  ]  
}
```

根据 JSON 数据包创建实体类 WaitCaseList,如代码清单 5-34 所示。

代码清单 5-34

```
public class WaitCaseList:ErrorEntity  
{
```



```

public List<WaitCaseInfo> sessionlist { get; set; }
}

public class WaitCaseInfo
{
    /// <summary>
    /// 会话接入的时间, UNIX 时间戳
    /// </summary>
    public int createtime
    {
        get { return Utils.ConvertDateTimeInt(CreateTime); }
        set { CreateTime = Utils.UnixTimeToTime(value.ToString()); }
    }
    /// <summary>
    /// 用户来访时间, UNIX 时间戳
    /// </summary>
    public DateTime CreateTime { get; set; }
    public string openid { get; set; }
    /// <summary>
    /// 指定接待的客服, 为空则表示未指定客服
    /// </summary>
    public string kf_account { get; set; }
}

```

创建实体后, 在 `CustomerServices` 类中添加方法 `GetWaitCaseList`。此方法的功能为获取未接入会话列表, 如代码清单 5-35 所示。

代码清单 5-35

```

public static WaitCaseList GetWaitCaseList(string accessToken)
{
    var url = string.Format("https://api.weixin.qq.com/customservice/kfsession/getwaitcase?access_token={0}", accessToken);
    return Utils.GetResult<WaitCaseList>(url);
}

```

5.5 PC 客户端自定义插件接口

在 PC 多客服客户端中, 微信公众平台预留了开放能力, 开发者可以在多客服聊天窗



口右侧区域添加自己的自定义 Web 页面插件（推荐页面宽度为 420px）。通过多客服客户端提供的 JavaScript 接口，开发者的页面可以与多客服客户端进行交互，实现自己需要的功能，如常用回复、信息查询等。

当需要添加自定义 Web 页面插件时，开发者需要先进入微信公众平台，依次进入“功能”→“多客服”→“客户端高级设置”→“添加插件”，如图 5-11 所示。输入页面地址和插件名称，然后重新登录多客服 PC 客户端，即可在聊天窗口右侧查看新添加的插件页面。官方提供了 demo，也可以先填写“demo”和 <http://dkf.qq.com/demopage.html>，使用官方 demo 进行体验。

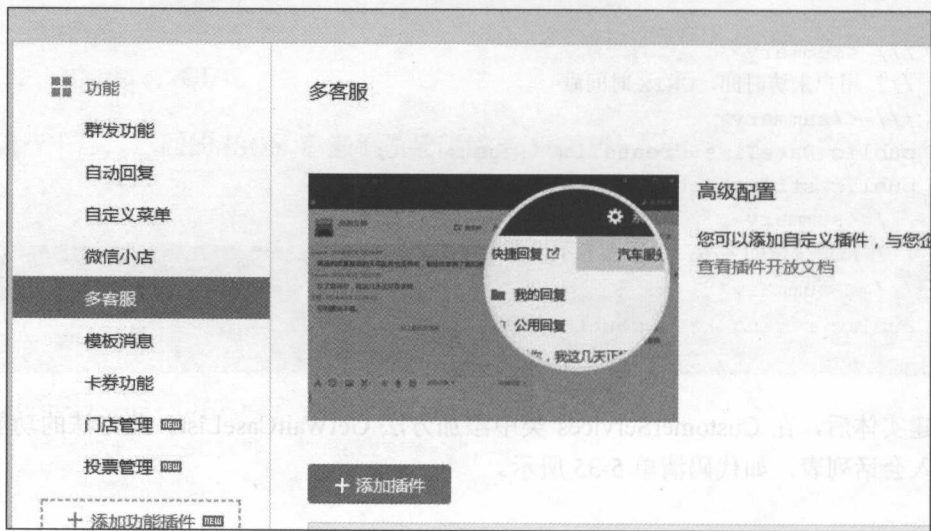


图 5-11

添加插件官方 demo 后，多客服客户端效果图如图 5-12 所示。

多客服客户端提供了一系列 JavaScript 方法接口，在多客服中加载的插件页面可以通过 JavaScript 调用客户端提供的方法。需要注意的是，多客服会给每个插件的 URL 添加一个 ticket 参数，ticket 用于加载多客服 JS 接口。下面的小节中将一一介绍这些接口。

5.5.1 接口调试

当需要调试自定义插件页面时，开发者可以使用浏览器的开发者工具进行调试。打开方式是切换到需要调试的插件页中，然后按 F12 键，此时将会打开计算机默认浏览器的开



发者工具。推荐使用 Chrome 浏览器，如图 5-13 所示。

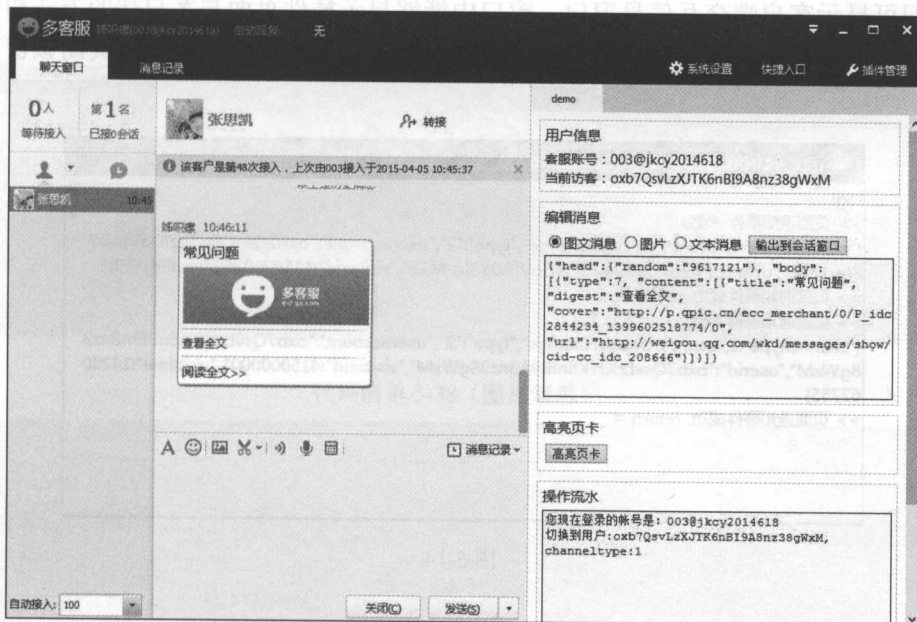


图 5-12

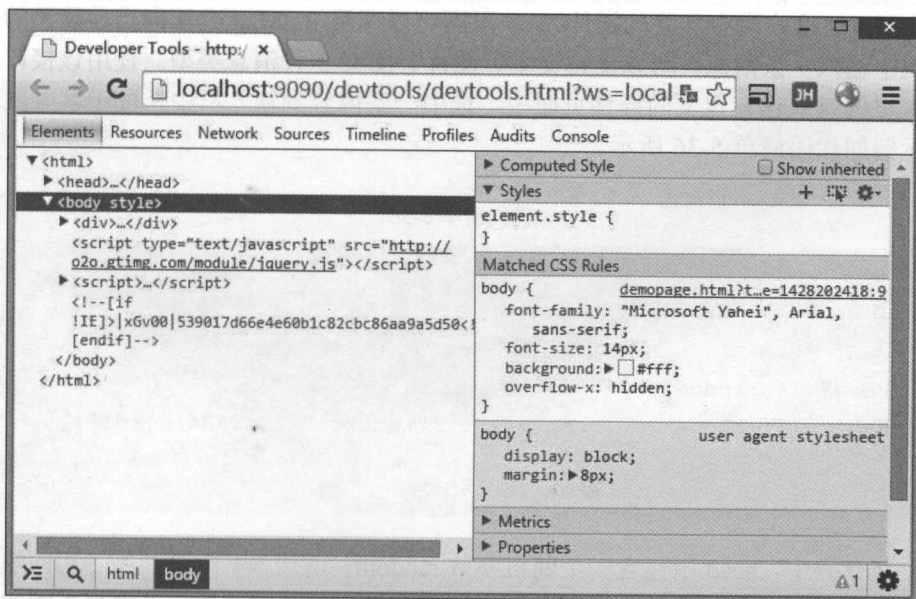


图 5-13



当需要在多客服 PC 客户端中调试时,你可以在多客服插件页面中按“Ctrl+Alt+0”组合键,即可显示客户端交互信息窗口。窗口中能够显示插件页面与客户端的方法和事件交互日志,方便开发者调试方法调用和事件通知。最后按“Alt+F4”组合键关闭该窗口,如图 5-14 所示。

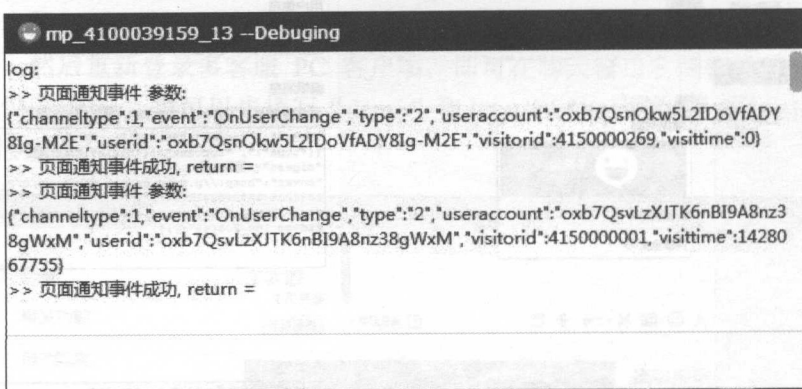


图 5-14

5.5.2 向会话窗口输入框中输入一条消息

多客服 PC 客户端中加载的插件 Web 页面可以调用 PutMsg 方法向当前聊天的会话窗口输入框中输入一条消息,可以支持文本、图片、图文 3 种消息类型。使用该接口,可以让客服人员快速从你自定义的插件页面中,选中一条内容到输入框中,便于客服发送。接口调用示例如代码清单 5-36 所示。

代码清单 5-36

```
window.external.PutMsg('{  
  "msg":  
  {  
    "head": {"random": "123456"},  
    "body": [{ "type": 0, "content": {"text": "234h2j34khjsdf"}}]  
  }  
})
```

在上述代码中, head 中包含一个随机参数 random。请传入一个随机数。相同随机数的调用会被抛弃,以避免被异常重复调用。body 中是消息的具体内容,各消息类型的参数定



义不同，具体消息所对应的 JSON 参数如代码清单 5-37 到代码清单 5-39 所示。

代码清单 5-37（文本消息）

```
{
  "type": 0,
  "content": {
    "text": "文本内容"
  }
}
```

代码清单 5-38（图片消息）

```
{
  "type": 1,
  "content": {
    "picUrl": "图片的 url"
  }
}
```

代码清单 5-39（图文消息）

```
{
  "type": 7,
  "content": [
    {
      "title": "标题",
      "digest": "摘要",
      "cover": "封面图片 url",
      "url": "要跳转的链接"
    },
    //多图文会有多项
  ]
}
```

为了方便调用，我们可以将多客服相关的 js 进一步封装，如代码清单 5-40 所示。



```
var Dkf = {  
    //初始化多客服  
    Init: function (ticket) {  
        $.getScript('http://crml.dkf.qq.com/php/index.php/thirdapp/appdemo/  
        get_workeraccount_by_sessionkey?callback=wokeraccountCallback&ticket='  
ticket);  
    },  
    //向会话窗口输入框中输入一条消息  
    PutMsg: function (type, content) {  
        var obj = {  
            msg: {  
                head: { random: Math.ceil(Math.random() * 10000000).toString() },  
                body: [{ type: type, content: content }]  
            }  
        };  
        window.external.PutMsg(JSON.stringify(obj));  
    },  
    //向会话窗口输入框中输入文本消息  
    PutTextMsg: function (text) {  
        var content = { text: text };  
        Dkf.PutMsg(0, content);  
    },  
    //向会话窗口输入框中输入图片消息  
    PutImgMsg: function (picUrl) {  
        var content = { picUrl: picUrl };  
        Dkf.PutMsg(1, content);  
    },  
    //向会话窗口输入框中输入图文消息  
    PutNewsMsg: function (news) {  
        Dkf.PutMsg(7, news);  
    }  
};
```

按照如图 5-15 所示的调用方式调用，当单击“输入消息”按钮时，将在聊天输入框中



输入一条图文、一条文本和一条图片消息，如图 5-16 所示。

```
<script src="http://libs.baidu.com/jquery/1.8.3/jquery.min.js"></script>
<script src="dkf.js"></script>
<script>
    $(function() {
        Dkf.Init("<%=Request.QueryString["ticket"]%>");
        $("#bb").click(function() {
            var dd = [
                {
                    title: "白哦天",
                    digest: "摘要",
                    cover: "http://vpvle.xicp.net/wx.jpg",
                    url: "http://www.baidu.com"
                }, {
                    title: "白哦12天",
                    digest: "摘要212",
                    cover: "http://vpvle.xicp.net/wx.jpg",
                    url: "http://www.baidu.com"
                }
            ];
            Dkf.PutNewsMsg(dd);
            Dkf.PutTextMsg("您好啊");
            Dkf.PutImgMsg("http://vpvle.xicp.net/wx.jpg");
        });
    });
</script>
```

图 5-15

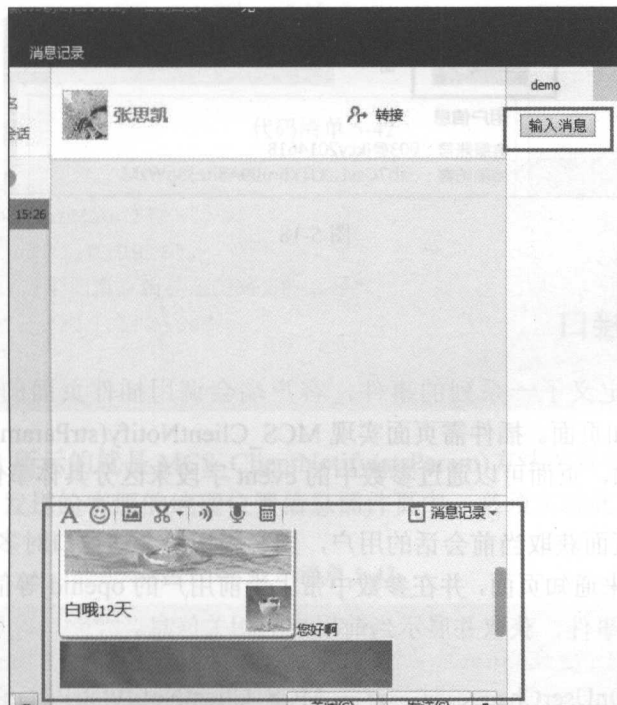


图 5-16



5.5.3 高亮自定义插件 Tab 页

多客服 PC 客户端以 Tab 页卡的方式显示多个自定义插件 Web 页面。客服可能会切换到其他插件页面。为了避免客服遗漏非当前显示的插件页面上的重要信息，插件页面可以调用 Notice 方法使自己的 Tab 页卡高亮，提示客服切换至你的页面查看信息。

为了方便调用，将方法添加到 Dkf 对象中，如图 5-17 所示。

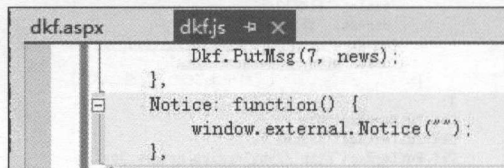


图 5-17

运行效果如图 5-18 所示。

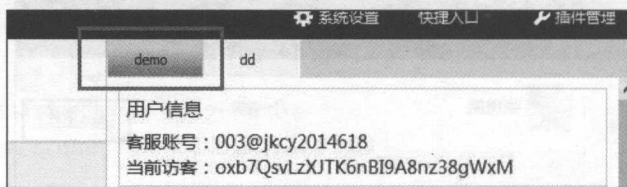


图 5-18

5.5.4 事件接口

多客服客户端定义了一系列的事件，客户端会调用插件页面的 MCS_ClientNotify(strParam)方法来通知页面。插件需页面实现 MCS_ClientNotify(strParam)方法，方可收到来自客户端的事件通知，页面可以通过参数中的 event 字段来区分具体事件类型。

为了方便插件页面获取当前会话的用户，当客服切换会话窗口时多客服客户端会通过 OnUserChange 事件来通知页面，并在参数中带上当前用户的 openid 等信息。插件页面可以响应 OnUserChange 事件，获取并展示当前用户的相关信息。

当事件类型为 OnUserChange 时，方法 MCS_ClientNotify(strParam)的参数格式如代码清单 5-41 所示。



代码清单 5-41

```
{
  "channeltype": 1,
  "event": "OnUserChange",
  "type": "2",
  "useraccount": "0xb7QsnOkw5L2IDoVfADY8Ig-M2E",
  "userid": "0xb7QsnOkw5L2IDoVfADY8Ig-M2E",
  "visitorid": 4150000269,
  "visittime": 0
}
```

上述清单中的 useraccount 即当前用户的 openid。

另外,当客服人员单击用户发送过来的地理位置消息时,客户端会通过 OnMapMsgClick 事件通知页面,并在参数中带上经纬度等信息。插件页面可以响应 OnMapMsgClick 事件,为用户提供基于地理信息的服务。

当事件类型为 OnMapMsgClick 时,方法 MCS_ClientNotify(strParam)的参数格式如代码清单 5-42 所示。

代码清单 5-42

```
{
  "event": "OnMapMsgClick",
  "latitude": "31.020916",
  "location": "荆门市京山县东门路 58-2 号",
  "longitude": "113.112564",
  "scale": "16"
}
```

代码清单 5-43 所示的就是 MCS_ClientNotify(strParam)方法的具体实现。在需要获取当前用户或查看用户发送给客服的地理位置信息插件页中,必须实现此方法。

代码清单 5-43

```
function MCS_ClientNotify(EventData) {
  var edjson = (new Function("return " + EventData))();
  switch (edjson['event']) {
    case 'OnUserChange': {
```



```
        OnUserChange(edjson);
        break;
    }
    case 'OnMapMsgClick': {
        OnMapMsgClick(EventData);
        break;
    }
}

function OnUserChange(EventData) {
    var openid = EventData.useraccount;
}

function OnMapMsgClick(EventData) {
    var latitude = EventData.latitude;
    var longitude = EventData.longitude;
    var location = EventData.location;
    var scale = EventData.scale;
}
```


第 6 章 微信 JS-SDK

微信 JS-SDK 是微信公众平台面向网页开发者提供的基于微信内的网页开发工具包。通过使用微信 JS-SDK，网页开发者可借助微信高效地使用拍照、选图、语音、位置等手机系统的能力，同时可以直接使用微信分享、扫一扫、卡券、支付等微信特有的能力，为微信用户提供更优质的网页体验。

6.1 JS-SDK 使用步骤

步骤一：登录微信公众平台，依次进入“公众号设置”→“功能设置”→“JS 接口安全域名”，设置 JS 接口安全域名后，公众号开发者可在该域名下调用微信开放的 JS 接口。填写的域名要求是一级或一级以上域名，需通过 ICP 备案的验证。可以填写 3 个域名，如图 6-1 所示。

在开发的过程中，开发者可以使用微信测试号，测试号可设置一个安全域名，如图 6-2 所示。

步骤二：在需要调用 JS 接口的页面引入如下 JS 文件（支持 https）：

<http://res.wx.qq.com/open/js/jweixin-1.0.0.js>

步骤三：通过 config 接口注入权限验证配置，如代码清单 6-1 所示。所有需要使用 JS-SDK 的页面必须先注入配置信息；否则将无法调用（同一个 URL 仅需调用一次。对于变化 URL 的 SPA 的 Web App，可在每次 URL 变化时进行调用。目前 Android 微信客户端



不支持 pushState 的 H5 新特性，所以使用 pushState 来实现 Web App 的页面会导致签名失败，此问题会在 Android 6.2 中修复）。

JS接口安全域名

设置JS接口安全域名后，公众号开发者可在该域名下调用微信开放的JS接口。
填写的JS接口安全域名要求是一级或一级以上域名，须通过ICP备案的验证，可填写三个域名(例：qq.com)。

域名1

域名2

域名3

确定 关闭

图 6-1

JS接口安全域名修改

设置JS接口安全域后，通过关注该测试号，开发者即可在该域名下调用微信开放的JS接口，请阅读微信JSSDK开发文档。

域名 ypile.xicp.net

图 6-2

代码清单 6-1

```
wx.config({
  debug: true, //开启调试模式，调用的所有 api 的返回值会在客户端 alert 出来。若要
               //查看传入的参数，则可以在 PC 端打开。参数信息会通过 log 打出，仅在
               //PC 端时才会打印
  appId: '', //必填，公众号的唯一标识
  timestamp: , //必填，生成签名的时间戳
```



```
nonceStr: '', //必填, 生成签名的随机串
signature: '', //必填, 签名
jsApiList: [] //必填, 需要使用的 JS 接口列表
});
```

在上述代码中, `signature` 表示的是签名。生成签名之前必须先了解一下 `jsapi_ticket`, `jsapi_ticket` 是公众号用于调用微信 JS 接口的临时票据。正常情况下, `jsapi_ticket` 的有效期为 7200 秒, 通过 `access_token` 来获取。由于获取 `jsapi_ticket` 的 API 调用次数非常有限, 频繁刷新 `jsapi_ticket` 会导致 API 调用受限, 影响自身业务, 因此开发者必须在自己的服务全局缓存 `jsapi_ticket`。`jsapi_ticket` 的缓存方式可参考 3.1.2 节 (获取 `access_token`) 中缓存 `access_token` 的处理方式, 在此不再赘述。

获取 `jsapi_ticket` 的接口地址如下:

`https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_token=ACCESS_TOKEN&type=jsapi`

HTTP 请求方式: GET。

成功调用后返回如代码清单 6-2 所示的 JSON 数据包。

代码清单 6-2

```
{
  "errcode": 0,
  "errmsg": "ok",
  "ticket": "bxLdikRXVbTPdHSM05e5u5sUoXNKd8-41ZO3MhKoyN5OfkWITDGgnr2fw
J0m9E8NYzWKVZvdV",
  "expires_in": 7200
}
```

根据上述代码清单创建实体类 `JsApiTicket`, 如代码清单 6-3 所示。

代码清单 6-3

```
public class JsApiTicket : ErrorEntity
{
    /// <summary>
    /// ticket
    /// </summary>
```



```
public string ticket { get; set; }
private int _expires_in;
/// <summary>
/// 有效期时间。单位为秒
/// </summary>
public int expires_in
{
    get { return _expires_in; }
    set
    {
        //获取失效时间
        expires_time = DateTime.Now.AddSeconds(value);
        _expires_in = value;
    }
}
/// <summary>
/// 失效时间
/// </summary>
public DateTime expires_time { get; set; }
}
```

在 WxApi 项目中添加 JsApi 类，用于封装 JS-SDK 相关的处理方法。在 JsApi 类中添加方法 GetHsJsApiTicket，方法的功能是根据 access_token 获取 jsapi_ticket，如代码清单 6-4 所示。

代码清单 6-4

```
public static JsApiTicket GetHsJsApiTicket(string accessToken)
{
    var url = string.Format("https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_token={0}&type=jsapi", accessToken);
    return Utils.GetResult<JsApiTicket>(url);
}
```

获取 jsapi_ticket 之后，就可以生成 JS-SDK 权限验证的签名了。签名生成规则如下：参与签名的字段包括 noncestr（随机字符串）、有效的 jsapi_ticket、timestamp（时间戳）、URL（当前网页的 URL，不包含#及其后面的部分）。对所有待签名参数按照字段名的 ASCII 码从小到大排序（字典序）后，使用 URL 键值对的格式（即 key1=val1&key2=val2……）



拼接成字符串 string1。这里需要注意的是所有参数名均为小写字母。对 string1 做 SHA1 加密，字段名和字段值都采用原始值，不进行 URL 转义。

注意事项：签名用的 noncestr 和 timestamp 必须与 wx.config 中的 nonceStr 和 timestamp 相同。签名用的 URL 必须是调用 JS 接口页面的完整 URL。出于安全考虑，开发者必须在服务端实现签名的逻辑。

根据签名算法的逻辑，在 JsApi 类中添加方法 GetJsApiSign，用于生成签名，如代码清单 6-5 所示。

代码清单 6-5

```
public static string GetJsApiSign(string noncestr, string jsapi_ticket,
string timestamp, string url)
{
    //将字段添加到列表中
    string[] arr = new[]
    {
        string.Format("noncestr={0}", noncestr),
        string.Format("jsapi_ticket={0}", jsapi_ticket),
        string.Format("timestamp={0}", timestamp),
        string.Format("url={0}", url)
    };
    //字典排序
    Array.Sort(arr);
    //使用 URL 键值对的格式拼接成字符串
    var temp = string.Join("&", arr);
    return FormsAuthentication.HashPasswordForStoringInConfigFile(temp,
"SHA1");
}
```

代码清单 6-1 中的 jsApiList 指的是当前页面需要使用的 JS 接口列表。所有 JS 接口列表如表 6-1 所示。

表 6-1

接 口	说 明
onMenuShareTimeline	获取“分享到朋友圈”按钮单击状态及自定义分享内容接口
onMenuShareAppMessage	获取“分享给朋友”按钮单击状态及自定义分享内容接口
onMenuShareQQ	获取“分享到 QQ”按钮单击状态及自定义分享内容接口
onMenuShareWeibo	获取“分享到腾讯微博”按钮单击状态及自定义分享内容接口



续表

接 口	说 明
startRecord	开始录音接口
stopRecord	停止录音接口
onVoiceRecordEnd	监听录音自动停止接口
playVoice	播放语音接口
pauseVoice	暂停播放接口
stopVoice	停止播放接口
onVoicePlayEnd	监听语音播放完毕接口
uploadVoice	上传语音接口
downloadVoice	下载语音接口
chooseImage	拍照或从手机相册中选图接口
previewImage	预览图片接口
uploadImage	上传图片接口
downloadImage	下载图片接口
translateVoice	识别音频并返回识别结果接口
getNetworkType	获取网络状态接口
openLocation	使用微信内置地图查看位置接口
getLocation	获取地理位置接口
hideOptionMenu	隐藏右上角菜单接口
showOptionMenu	显示右上角菜单接口
hideMenuItems	批量隐藏功能按钮接口
showMenuItems	批量显示功能按钮接口
hideAllNonBaseMenuItem	隐藏所有非基础按钮接口
showAllNonBaseMenuItem	显示所有功能按钮接口
closeWindow	关闭当前网页窗口接口
scanQRCode	调起微信扫一扫接口
chooseWXPay	发起一个微信支付请求
openProductSpecificView	跳转微信商品页接口
addCard	批量添加卡券接口
chooseCard	调起适用于门店的卡券列表并获取用户选择列表
openCard	查看微信卡包中的卡券接口

步骤四：通过 ready 接口处理成功验证。

```
wx.ready(function(){
//config 信息验证后会执行 ready 方法。所有接口调用都必须在 config 接口获得结果之后
// config 是一个客户端的异步操作，所以如果需要在页面加载时就调用相关接口，则须把相关接口
//放在 ready 函数中调用来确保正确执行。对于用户触发时才调用的接口，则可以直接调用，无须放
//在 ready 函数中
});
```



步骤五：通过 error 接口处理失败验证。

```
wx.error(function(res) {  
    //config 信息验证失败会执行 error 函数。如签名过期导致验证失败，则具体错误信息可以  
    //打开 config 的 debug 模式查看，也可以在返回的 res 参数中查看，对于 SPA 可以在这里更  
    //新签名  
});
```

请注意，所有接口通过 wx 对象（也可以使用 jWeixin 对象）来调用。参数是一个对象，除了每个接口本身需要传的参数之外，还有以下通用参数。

- success: 接口调用成功时执行的回调函数。
- fail: 接口调用失败时执行的回调函数。
- complete: 接口调用完成时执行的回调函数，无论成功或失败都会执行。
- cancel: 用户单击取消时的回调函数，仅部分有用户取消操作的 API 才会用到。
- trigger: 监听 Menu 中的按钮单击时触发的方法，该方法仅支持 Menu 中的相关接口（不要尝试在 trigger 中使用 ajax 异步请求修改本次分享的内容，因为客户端分享操作是一个同步操作，这时候使用 ajax 的回包还没有返回）。

以上几个函数都带有一个参数，类型为对象，其中除了每个接口本身返回的数据之外，还有一个通用属性 errMsg，其值格式如下。

- 调用成功时: "xxx:ok"，其中 xxx 为调用的接口名。
- 用户取消时: "xxx:cancel"，其中 xxx 为调用的接口名。
- 调用失败时: 其值为具体错误信息。

步骤六：判断当前客户端版本是否支持指定的 JS 接口。

在开发的过程中使用 JS-SDK 时，在排错的过程中，可以先判断当前客户端是否支持当前使用的 JS 接口，如代码清单 6-6 所示。

代码清单 6-6

```
wx.checkJsApi({  
    jsApiList: ['chooseImage'], // 需要检测的 JS 接口列表
```



```
success: function(res) {  
    // 以键值对的形式返回, 可用的 API 值为 true, 不可用为 false  
    // 如: {"checkResult":{"chooseImage":true},"errMsg":"checkJsApi:ok"}  
});
```

下面将以实例的方式讲解各个接口的使用。首先, 在项目中添加页面 WxJs.aspx, 在页面的 cs 文件中添加如代码清单 6-7 所示的代码。

代码清单 6-7

```
protected string timestamp;//时间戳  
protected string noncestr;//随机字符串  
protected string url;//当前 URL  
protected string sign;//签名  
private static WxApi.ReceiveEntity.JsApiTicket ticket;  
protected void Page_Load(object sender, EventArgs e)  
{  
    timestamp = Utils.ConvertDateTimeInt(DateTime.Now).ToString();  
    noncestr = timestamp;//随机字符串也使用时间戳  
    url = "http://" + Utils.GetCurrentFullHost() + Request.RawUrl;  
    var accessToken = AccessTokenBox.GetTokenValue("your appid", "your  
appsecret");  
    if (ticket == null || ticket.expires_time < DateTime.Now)  
    {  
        ticket = JsApi.GetHsJsApiTicket(accessToken);  
    }  
    sign = JsApi.GetJsApiSign(noncestr, ticket.ticket, timestamp, url);  
}
```

在上述代码清单中, Utils.GetCurrentFullHost 方法的功能是获取当前请求的完整主机头, 代码如下所示:

```
/// <summary>  
/// 得到当前完整主机头  
/// </summary>  
/// <returns></returns>  
public static string GetCurrentFullHost()  
{  
    HttpRequest request = System.Web.HttpContext.Current.Request;  
    if (!request.Url.IsDefaultPort)  
        return string.Format("{0}:{1}",  
            request.Url.Host, request.Url.Port.ToString());  
    return request.Url.Host;  
}
```




然后根据前面所讲的接入步骤，在页面的前台引入 js，通过 config 接口注入权限验证配置，如图 6-3 所示。

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.8.2.min.js"></script>
<script src="http://res.wx.qq.com/open/js/jweixin-1.0.0.js"></script>
<script type="text/javascript">
  wx.config({
    debug: false, // 开启调试模式,调用的所有api的返回值会在客户端alert出来,若要
    appId: 'wx7008116979a800d', // 必填,公众号的唯一标识
    timestamp: '<%=timestamp%>', // 必填,生成签名的时间戳
    nonceStr: '<%=noncestr%>', // 必填,生成签名的随机串
    signature: '<%=sign%>', // 必填,签名
    jsApiList: [ 'checkJsApi' ] // 必填,需要使用的JS接口列表,所有JS接口列表见
  });
```

图 6-3

在页面中添加一个按钮“checkJsApi”，在 ready 处理成功后，给按钮注册单击事件，用于检测当前客户端是否支持某个接口，如图 6-4 所示。

此时，当在微信中访问此页面，并单击“checkJsApi”按钮时，将会弹出检测结果，如图 6-5 所示。

```
wx.ready(function() {
  $('#checkJsApi').click(function() {
    wx.checkJsApi({
      jsApiList: [
        'getNetworkType',
        'previewImage'
      ],
      success: function(res) {
        alert(JSON.stringify(res));
      }
    });
  });
});
```

图 6-4

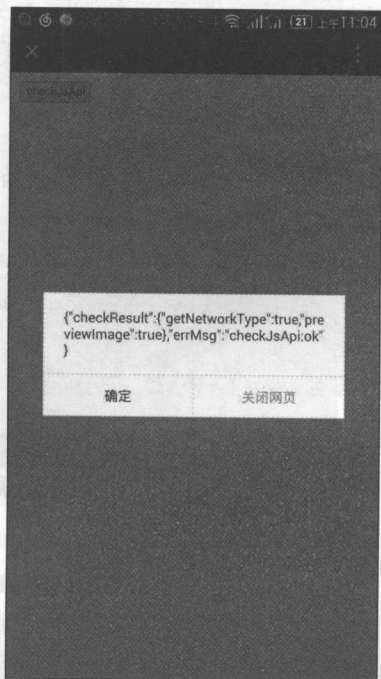


图 6-5



6.2 分享接口

分享接口包括分享到朋友圈、分享给朋友、分享到 QQ、分享到腾讯微博。使用分享接口可以自定义分享的内容，包括分享的标题、链接、图标等。

当用户分享没有注册分享接口的页面时，微信客户端默认使用网页的 title 作为分享的标题，当前页面的 URL 作为分享的 URL，页面中的第一张图片作为分享图标。而大多数的时候，开发者需要自定义用户的分享内容，此时分享接口就可以派上用场了。

在页面的 ready 接口中，添加如代码清单 6-8 所示的代码。

代码清单 6-8（分享到朋友圈）

```
wx.onMenuShareTimeline({
  title: '我要分享到朋友圈', //分享标题
  link: 'http://www.baidu.com', //分享链接
  imgUrl: 'http://ypyle.xicp.net/wx.jpg', //分享图标
  success: function () {
    alert("分享成功");
    //用户确认分享后执行的回调函数
  },
  cancel: function () {
    alert("取消分享了");
    //用户取消分享后执行的回调函数
  }
});
```

此时，当分享此页面到朋友圈时，分享的内容如图 6-6 左边所示。图 6-6 右边表示的是用户取消了分享后，执行取消分享回调后的效果。

其他分享接口和分享到朋友圈类似，如代码清单 6-9 到代码清单 6-11 所示。

代码清单 6-9（分享给朋友）

```
wx.onMenuShareTimeline({
  title: '我要分享给朋友', //分享标题
  link: 'http://www.baidu.com', //分享链接
```



```

imgUrl: 'http://ypyle.xicp.net/wx.jpg', //分享图标
success: function () {
    alert("分享成功");
    //用户确认分享后执行的回调函数
},
cancel: function () {
    alert("取消分享了");
    //用户取消分享后执行的回调函数
}
});

```

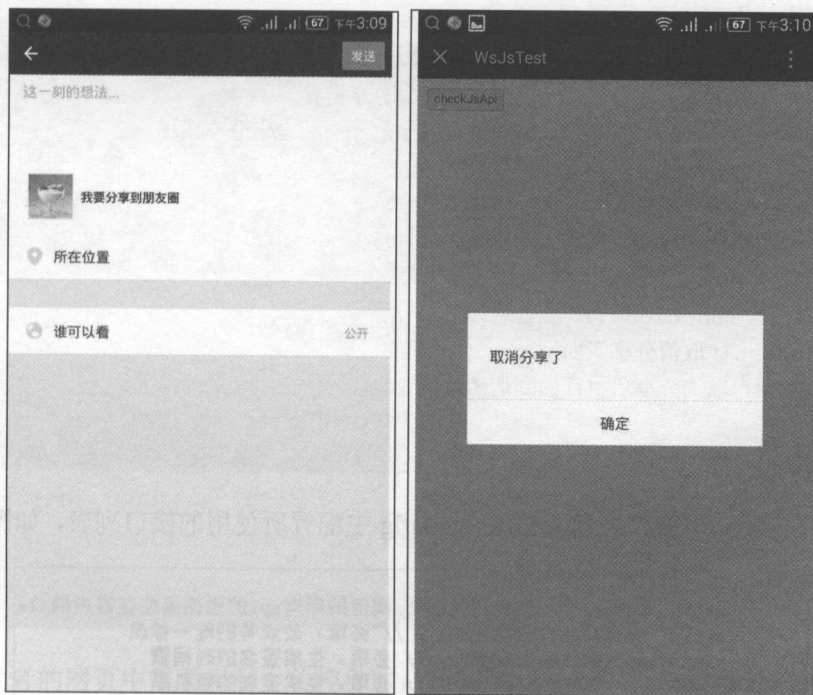


图 6-6

代码清单 6-10 (分享到 QQ)

```

wx.onMenuShareQQ ({
    title: '我要分享到 QQ', //分享标题
    link: 'http://www.baidu.com', //分享链接
    imgUrl: 'http://ypyle.xicp.net/wx.jpg', //分享图标

```



```
success: function () {  
    alert("分享成功");  
    //用户确认分享后执行的回调函数  
},  
cancel: function () {  
    alert("取消分享了");  
    //用户取消分享后执行的回调函数  
}  
});
```

代码清单 6-11 (分享到腾讯微博)

```
wx.onMenuShareWeibo ({  
    title: '我要分享到腾讯微博', //分享标题  
    link: 'http://www.baidu.com', //分享链接  
    imgUrl: 'http://ypyle.xicp.net/wx.jpg', //分享图标  
    success: function () {  
        alert("分享成功");  
        //用户确认分享后执行的回调函数  
    },  
    cancel: function () {  
        alert("取消分享了");  
        //用户取消分享后执行的回调函数  
    }  
});
```

注意: 在使用任何接口之前, 必须在 config 中配置所使用的接口列表, 如图 6-7 所示。

```
wx.config({  
    debug: false, // 开启调试模式,调用的所有api的返回值会在客户端ale  
    appId: 'wxd7008116979a800d', // 必填, 公众号的唯一标识  
    timestamp: '<%=timestamp%>', // 必填, 生成签名的时间戳  
    nonceStr: '<%=noncestr%>', // 必填, 生成签名的随机串  
    signature: '<%=sign%>', // 必填, 签名  
    jsApiList: [  
        'checkJsApi',  
        'onMenuShareTimeline',  
        'onMenuShareAppMessage',  
        'onMenuShareWeibo',  
        'onMenuShareQQ'  
    ] // 必填, 需要使用的JS接口列表
```

图 6-7



6.3 图像接口

图像接口包括拍照或从手机相册选图、预览图片、上传图片 and 下载图片。拍照或从手机相册选图功能类似于 html 的上传控件，具体实现如代码清单 6-12 和代码清单 6-13 所示。

代码清单 6-12 (拍照或从手机相册中选图接口)

```
wx.chooseImage({
  success: function (res) {
    var localIds = res.localIds; //返回选定照片的本地 ID 列表，localId 可以
                                //作为 img 标签的 src 属性显示图片。在使用的
                                //过程中，可以遍历 localIds，然后再调用
                                //上传图像接口，将图片分别上传到微信服务器
  }
});
```

代码清单 6-13 (上传图片接口)

```
wx.uploadImage({
  localId: '', //需要上传的图片的本地 ID，由 chooseImage 接口获得
  isShowProgressTips: 1, //默认为 1，显示进度提示
  success: function (res) {
    var serverId = res.serverId; //返回图片的服务器端 ID
  }
});
```

当开发者的网页中需要用到上传控件时，**chooseImage** 和 **uploadImage** 结合使用可完美实现上传控件的功能。需要注意的是，**uploadImage** 接口是将图片上传到微信的服务器，和临时素材一样，有效期为 3 天，上传完毕后，获取的 **serverId** 即上传临时素材接口返回的 **media_id**。开发者如需将上传的图片保存在自己的服务器，则可以在上传完成后，调用下载临时素材接口。或者调用 JS 的下载接口，将图片下载到用户的客户端。下载到用户客户端的图片返回的是 **localId**，此处的 **localId** 和 **chooseImage** 接口的 **localId** 是一样的，可以在 html 的 **img** 标签中显示，如代码清单 6-14 所示。



代码清单 6-14（下载图片接口）

```
wx.downloadImage({
  serverId: '', //需要下载的图片的服务器端 ID，由 uploadImage 接口获得
  isShowProgressTips: 1, //默认为 1，显示进度提示
  success: function (res) {
    var localId = res.localId; //返回图片下载后的本地 ID
  }
});
```

在 Web 开发的过程中，有时需要实现类似于相册的功能，JS-SDK 中的 **previewImage** 可完美实现此功能。如代码清单 6-15 所示，此接口有两个参数，**current** 表示的是当前显示的图片链接；**urls** 是个数组，表示的是需要预览的图片链接列表。

代码清单 6-15（预览图片接口）

```
wx.previewImage({
  current: '', //当前显示的图片链接
  urls: [] //需要预览的图片链接列表
});
```

注意：参数中的图片链接必须是网络路径，如 `http://www.xx.xx.jpg`，不能是服务器的相对路径，如 `“/image/xx.jpg”`；否则相对路径图片将无法加载。

6.4 音频接口

音频接口包括开始录音、停止录音、监听录音自动停止、播放语音、暂停播放、停止播放、监听语音播放完毕、上传语音、下载语音以及识别音频并返回识别结果。本节将通过一个完整的实例来分别讲述。

首先，在 **config** 中配置音频相关的接口权限，如图 6-8 所示。

然后在页面中添加如图 6-9 所示的按钮，用于控制录音、播放录音以及上传、下载。

当单击“开始录音”按钮时，调用录音接口（第一次使用此网页录音时，会询问是否允许录音），即可开始录音，如代码清单 6-16 所示。



```
wx.config({
  debug: true, // 开启调试模式,调用的所有api的返回值会在console打印,请勿视为bug
  appId: 'wx7008116979a800d', // 必填,公众号的唯一标识
  timestamp: '<%=timestamp%>', // 必填,生成签名的时间戳
  nonceStr: '<%=noncestr%>', // 必填,生成签名的随机串
  signature: '<%=sign%>', // 必填,签名
  jsApiList: [
    'startRecord', //开始录音
    'onVoiceRecordEnd', //监听录音自动停止
    'stopRecord', //停止录音
    'playVoice', //播放语音
    'pauseVoice', //暂停播放
    'stopVoice', //停止播放
    'onVoicePlayEnd', //监听语音播放完毕
    'uploadVoice', //上传语音
    'downloadVoice', //下载语音
    'translateVoice' //识别音频
  ]
});
```

图 6-8

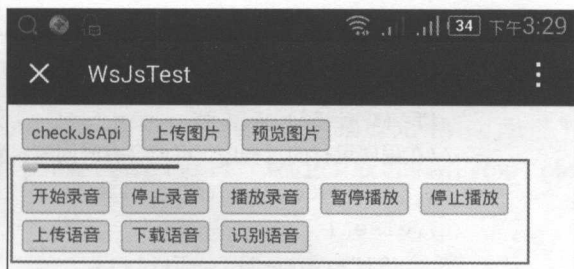


图 6-9

代码清单 6-16 (开始接口)

```
var voiceLocalId;
$("#startRecord").click(function() {
  i = 0;
  wx.startRecord({
    complete :function(res) {
      wx.onVoiceRecordEnd({
        // 录音时间超过 1 分钟没有停止的时候会执行 complete 回调
        complete: function (res1) {
          voiceLocalId = res1.localId;
        }
      });
      t = setInterval(rangestart, 1000);
    },
    cancel:function(res) {
      alert("您取消了录音");
    }
  });
});
```

startRecord 接口可以录制的音频长度最大为 60s, 当录音时间超过 60s 时, 就无法调用停止录音的接口。所以在调用录音接口时, 或者在 ready 事件中调用 onVoiceRecordEnd, 录音时间超过 1 分钟没有停止时会执行 complete 回调。开发者可在回调中停止录音。请注意: onVoiceRecordEnd 兼容性略差, 在部分安卓设备中是无效的。所以在上述代码中设



置了一个定时器，当开始录音时，同时启动一个定时器，当录音时间达到 60s 时，则调用“停止录音”按钮。定时器执行的代码如代码清单 6-17 所示。

代码清单 6-17

```
var rangestart= function() {  
    if (i ==55) {  
        //在程序运行的过程中，可能会有时间差。为了保险起见，这里设置了录音的最大时长为 55s  
        $("#stopRecord").click();  
    } else {  
        $("#range").val(i++);  
        $("#time").text(i);  
    }  
}
```

“停止录音”按钮的代码如代码清单 6-18 所示。

代码清单 6-18

```
$("#stopRecord").click(function() {  
    wx.stopRecord({  
        success: function(res) {  
            voiceLocalId = res.localId;  
            clearInterval(t); //清除计时器  
        }  
    });  
});
```

停止录音后，将 localId 保存在全局的 voiceLocalId 中，此时可以调用播放录音接口；或者停止录音后，就直接播放录音，如代码清单 6-19 所示。

代码清单 6-19（播放录音）

```
$("#playRecord").click(function() {  
    wx.playVoice({  
        localId:voiceLocalId //需要播放的音频的本地 ID  
    });  
    //监听语音播放完毕接口  
    wx.onVoicePlayEnd({  
        success: function (res) {
```




```
var localId = res.localId; //返回音频的本地 ID
alert("语音播放完毕");
}
});
});
```

和录音一样，可以使用 **onVoicePlayEnd** 接口监听语音播放。当播放完毕后，会执行 **success** 回调。在播放的过程中，也可以调用 **pauseVoice** 接口暂停播放，或者调用 **stopVoice** 接口停止播放，如代码清单 6-20 所示。

代码清单 6-20（停止或暂停播放）

```
$("#pauseVoice").click(function() {
    wx.pauseVoice({
        localId:voiceLocalId //需要暂停的音频的本地 ID
    });
});
$("#stopVoice").click(function() {
    wx.stopVoice({
        localId:voiceLocalId//需要停止的音频的本地 ID
    });
});
```

和图片接口一样，录音结束后，会生成一个本地 ID。开发者可以调用上传接口将录音文件上传到微信服务器，上传成功后，会返回一个 **media_id**，此 **media_id** 的有效期为 3 天。如果开发者需要将此文件保存在自己的服务器，则可以调用获取临时素材接口，将文件下载后保存在自己的服务器。上传录音的代码如代码清单 6-21 所示。

代码清单 6-21（上传录音）

```
$("#uploadVoice").click(function() {
    wx.uploadVoice({
        localId:voiceLocalId , //需要上传的音频的本地 ID，由 stopRecord 接口获得
        isShowProgressTips: 1, //默认为 1，显示进度提示
        success: function (res) {
            voiceServerId = res.serverId; //返回音频的服务器端 ID
        }
    });
});
```



根据 media_id 下载录音文件的代码如代码清单 6-22 所示。

代码清单 6-22 (下载录音)

```
$("#downloadVoice").click(function() {  
    wx.downloadVoice({  
        serverId: voiceServerId, //需要下载的音频的服务器端 ID, 由 uploadVoice 接口获得  
        isShowProgressTips: 1, //默认为 1, 显示进度提示  
        success: function(res) {  
            var localId = res.localId; //返回音频的本地 ID  
        }  
    });  
});
```

录音后, 还可以使用识别音频接口, 将语音识别为文字, 调用成功后将会返回识别结果, 如代码清单 6-23 所示。

代码清单 6-23 (语音识别为文字)

```
$("#translateVoice").click(function() {  
    wx.translateVoice({  
        localId: voiceLocalId, //需要识别的音频的本地 ID, 由录音相关接口获得  
        isShowProgressTips: 1, //默认为 1, 显示进度提示  
        success: function (res) {  
            alert(res.translateResult); //语音识别的结果  
        }  
    });  
});
```

6.5 地理位置

地理位置接口包括获取地理位置和使用微信内置地图查看位置接口。首先, 开发者需要知道的是, 地理位置的表达方式是坐标的形式, 但坐标又分为不同的坐标类型, 具体如下所示。

- GPS 角度坐标。
- GPS 米制坐标 (sogou 地图坐标)。
- Google 地图、soso 地图 (腾讯地图)、aliyun 地图、mapabc 地图和 amap 地图所用



坐标。

- Google 地图、soso 地图（腾讯地图）、aliyun 地图、mapabc 地图和 amap 地图所用的米制坐标。
- 百度地图采用的经纬度坐标。
- 百度地图采用的米制坐标。
- mapbar 地图坐标。
- 51 地图坐标。

在微信中，获取用户地理位置的方式有两种。一种是用户主动发送地理位置，如图 6-10 所示。使用此种方式发送的地理位置的坐标和谷歌坐标的类型是一致的。



图 6-10



另外一种方式是开通了获取用户地理位置接口的公众号，在每次用户和公众号进行对话时都会上报一次，或者用户进行对话后每隔 5s 上报一次。开发者可以自行选择获取地理位置的模式，如图 6-11 所示。

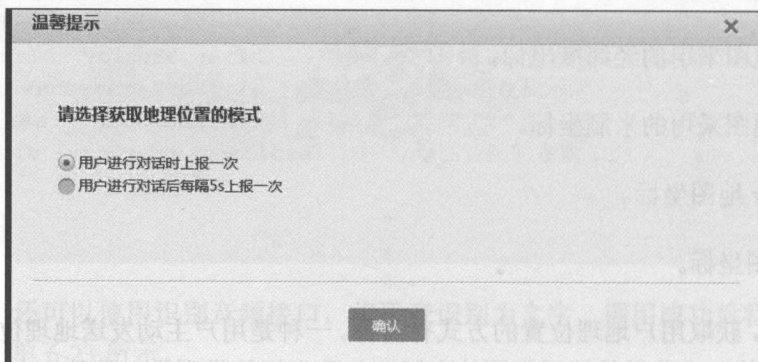


图 6-11

开通此功能后，用户在和公众号对话时，微信客户端会获取当前用户的地理位置并推送给开发者的服务器，推送的 XML 数据包如代码清单 6-24 所示。

代码清单 6-24

```
<xml>
  <ToUserName><![CDATA[gh_8dlff4fd69cd]]></ToUserName>
  <FromUserName><![CDATA[o8mXItyyhhtlubxsO9nidRzfMIOU]]></FromUserName>
  <CreateTime>1428555717</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[LOCATION]]></Event>
  <Latitude>30.528996</Latitude>
  <Longitude>114.355186</Longitude>
  <Precision>30.000000</Precision>
</xml>
```

至于处理此消息的方法请读者参考 3.3 节中的相关内容，在此不再赘述。这里需要特别说明的是，此消息的地理位置是通过微信客户端调用终端（手机、Pad 等）的 GPS 模块获取到的，并没有像用户主动发送地理位置那样通过地图进行定位，所以此种方式获取到的地理位置的坐标类型是没有经过处理的 GPS 角度坐标。同理，通过 JS-SDK 的获取地理位置接口获取的经纬度也是 GPS 角度坐标。代码清单 6-25 所示的就是使用 JS-SDK 的获取地理位置接口后，将获取到的 GPS 坐标转换成腾讯地图坐标。



代码清单 6-25

```
wx.getLocation({
  success: function (res) {
    latitude = res.latitude; //纬度，浮点数，范围为 90~-90
    longitude = res.longitude; //经度，浮点数，范围为 180~-180
    var speed = res.speed; //速度，以米/每秒计
    var accuracy = res.accuracy; //位置精度
    qq.maps.convertor.translate(new qq.maps.LatLng(latitude, longitude),
1, function(res){
      alert(res[0]);
    });
  }
});
```

使用微信内置地图查看位置接口的代码如代码清单 6-26 所示。

代码清单 6-26

```
wx.openLocation({
  latitude: 0, //纬度，浮点数，范围为 90~-90
  longitude: 0, //经度，浮点数，范围为 180~-180
  name: '', //位置名
  address: '', //地址详情说明
  scale: 1, //地图缩放级别，整型值，范围为 1~28。默认为最大
  infoUrl: '' //在查看位置界面底部显示的超链接，可单击跳转
});
```

由于使用不同类型的坐标产生的误差是很大的，而微信内置的地图是腾讯地图，因此开发者在调用的过程中应将坐标类型统一。

6.6 界面操作

界面操作主要是对微信内置浏览器右上角的菜单的操作，包括菜单的全部或部分的显示与隐藏及关闭当前网页窗口接口，如图 6-12 所示。

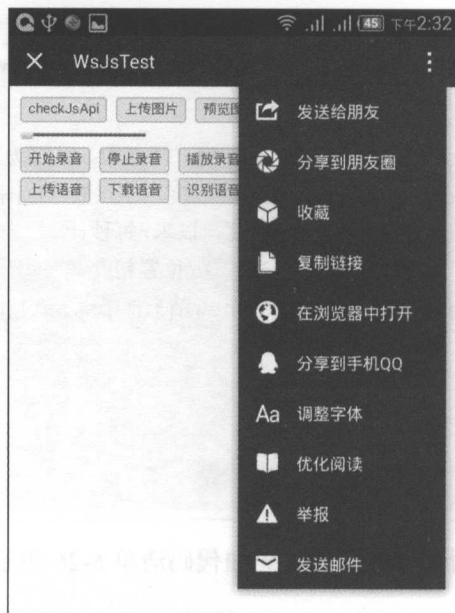


图 6-12

当执行 `wx.hideOptionsMenu()` 时，如图 6-12 所示的右上角的菜单将被隐藏，用户将无法使用分享、在浏览器中打开等功能，此接口主要用于有些不想让用户分享或查看网页链接的页面。当右上角菜单隐藏后，执行 `wx.showOptionsMenu()` 可以显示右上角菜单。开发者也可以有选择地批量操作部分功能按钮和非基础按钮。批量操作功能按钮的接口如代码清单 6-27 所示。

代码清单 6-27

```
wx.hideMenuItems({
  menuList: [] //要隐藏的菜单项，只能隐藏“传播类”和“保护类”按钮
});
wx.showMenuItems({
  menuList: [] //要显示的菜单项
});
```

批量隐藏所有非基础按钮接口和显示所有功能按钮接口的代码如代码清单 6-28 所示。

代码清单 6-28

```
wx.hideAllNonBaseMenuItem();//隐藏所有非基础按钮
```



```
wx.showAllNonBaseMenuItem();//显示所有功能按钮
```

所有菜单项列表如下所示。

基本类

- 举报: "menuItem:exposeArticle"
- 调整字体: "menuItem:setFont"
- 日间模式: "menuItem:dayMode"
- 夜间模式: "menuItem:nightMode"
- 刷新: "menuItem:refresh"
- 查看公众号 (已添加): "menuItem:profile"
- 查看公众号 (未添加): "menuItem:addContact"

传播类

- 发送给朋友: "menuItem:share:appMessage"
- 分享到朋友圈: "menuItem:share:timeline"
- 分享到 QQ: "menuItem:share:qq"
- 分享到 Weibo: "menuItem:share:weiboApp"
- 收藏: "menuItem:favorite"
- 分享到 FB: "menuItem:share:facebook"
- 分享到 QQ 空间: "menuItem:share:QZone"

保护类

- 调试: "menuItem:jsDebug"



- 编辑标签: "menuItem:editTag"
- 删除: "menuItem:delete"
- 复制链接: "menuItem:copyUrl"
- 原网页: "menuItem:originPage"
- 阅读模式: "menuItem:readMode"
- 在 QQ 浏览器中打开: "menuItem:openWithQQBrowser"
- 在 Safari 中打开: "menuItem:openWithSafari"
- 邮件: "menuItem:share:email"
- 一些特殊公众号: "menuItem:share:brand"

另外, 当需要关闭当前网页时, 可执行 `wx.closeWindow()` 命令。

6.7 微信扫一扫接口

此接口可以使用户在网页中调用扫一扫功能, 如代码清单 6-29 所示。

代码清单 6-29

```
wx.scanQRCode({
  needResult: 0, //默认为 0, 扫描结果由微信处理; 为 1 则直接返回扫描结果
  scanType: ["qrCode", "barCode"], //可以指定扫二维码还是一维码, 默认二者都有
  success: function (res) {
    var result = res.resultStr; //当 needResult 为 1 时, 扫码返回的结果
  }
});
```

在上述代码中, 参数 `needResult` 的值为 0 时, 扫描结果由微信处理; 为 1 则直接返回扫描结果。比如用户扫描的二维码的内容是一个链接, 当为 0 时, 扫描后会直接跳转到此链接; 为 1 时则不会跳转, 而是返回扫描结果。



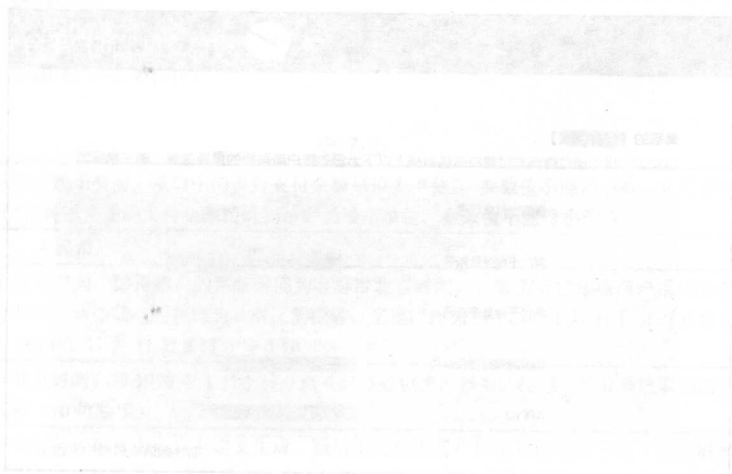
6.8 其他 JS 接口

在微网站开发的过程中，有的时候为了更好的用户体验，需要根据用户客户端当前的网络类型提供不同的展现方式。比如当用户访问一个在线订餐的微网站时，如果用户的网络环境是 3G 或者 2G，在这种环境下，可以将网站切换到无图模式，使用户能更快地访问网站。如果是在 4G 或 WiFi 的环境下，图文并茂的方式更容易勾起用户的食欲。代码清单 6-30 为获取网络状态的接口。

代码清单 6-30

```
wx.getNetworkType({
  success: function (res) {
    var networkType = res.networkType; //返回网络类型 2g、3g、4g、wifi
  }
});
```

除了前面讲到的接口外，JS-SDK 还提供了跳转微信商品页接口、微信卡券以及微信支付相关的接口，将在后续章节中详细讲解。



第7章 支付接口开发

7.1 微信支付简介

微信支付是基于微信客户端提供的支付服务功能，同时向商户提供销售经营分析、账户和资金管理的功能支持。提供多种支付方式，如刷卡支付、公众号支付、扫码支付等，以及代金券或立减优惠、现金红包、企业付款等多种支付工具。

商户在微信公众平台（申请扫描支付、公众号支付）或开放平台（申请 App 支付）按照相应提示，申请相应的微信支付模式。微信官方审核资料无误后开通相应的微信支付权限。微信支付申请审核通过后，商户在申请资料填写的邮箱中收取到由微信支付小助手发送的邮件，此邮件包含开发时需要使用的支付账户信息，如图 7-1 所示。

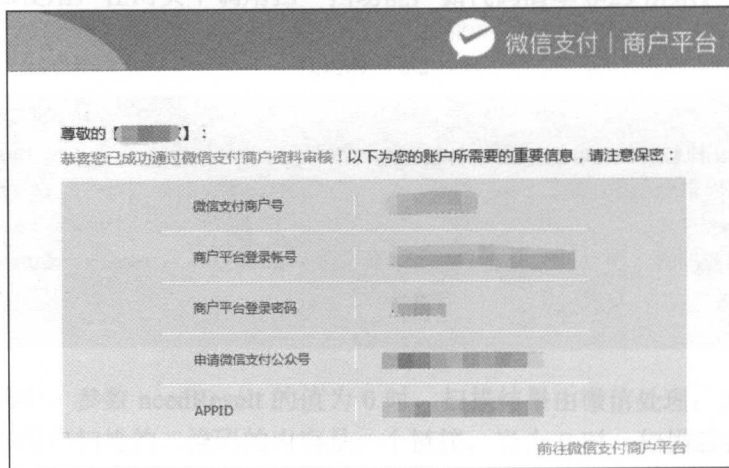


图 7-1



在开发者正式着手开发之前，需要前前往微信支付商户平台（<https://pay.weixin.qq.com>）。依次进入“账户设置”→“安全设置”→“API 安全”，安装操作证书后，设置支付密钥 key。

7.2 接口调用规则

7.2.1 协议规则

商户接入微信支付，为了账户交易安全，开发者调用 API 时必须遵循以下规则（见表 7-1）。

表 7-1

传输方式	为保证交易安全性，采用 HTTPS 传输
请求方式	采用 POST 方法提交
数据格式	提交和返回数据都为 XML 格式，根节点名为 XML
字符编码	统一采用 UTF-8 字符编码
签名算法	MD5
签名要求	请求和接收数据均需要校验签名
证书要求	调用申请退款、撤销订单接口需要商户证书
判断逻辑	先判断协议字段返回，再判断业务返回，最后判断交易状态

7.2.2 参数规定

具体参数规定如表 7-2 所示。

表 7-2

交易金额	默认为人民币交易，接口中的参数支付金额单位为“分”，参数值不能带小数。对账单中的交易金额单位为“元”。外币交易的支付金额精确到币种的最小单位， 参数值不能带小数点
货币类型	CNY：人民币
时间	标准北京时间。如果商户的系统时间为非标准北京时间，参数值必须根据商户系统所在时区先换算成标准北京时间，例如商户所在地为 0 时区的伦敦，当地时间为 2014 年 11 月 11 日 0 时 0 分 0 秒，换算成北京时间为 2014 年 11 月 11 日 8 时 0 分 0 秒
时间戳	标准北京时间。自 1970 年 1 月 1 日 0 点 0 分 0 秒以来的秒数。注意：部分系统取到的值为毫秒级，需要转换成秒（10 位数字）
商户订单号	商户支付的订单号由商户自定义生成，微信支付要求商户订单号保持唯一性（建议根据当前系统时间加随机序列来生成订单号）。重新发起一笔支付要使用原订单号，避免重复支付；已支付过或已调用关单、撤销的订单号不能重新发起支付



7.2.3 安全规范

安全规范包括签名的生成、随机数算法生成、证书调用以及商户回调 API 安全。

签名生成的通用步骤如下。

第一步，设所有发送或者接收到的数据为集合 M，将集合 M 内非空参数值的参数按照 ASCII 码从小到大排序（字典序），使用 URL 键值对的格式（即 key1=v1&key2=v2……）拼接成字符串 StringA。特别需要注意以下重要规则：

- 参数名 ASCII 码从小到大排序（字典序）。
- 如果参数的值为空，则不参与签名。
- 参数名区分大小写。
- 验证调用返回或微信主动通知签名时，传送的 sign 参数不参与签名，将生成的签名与该 sign 值进行校验。
- 微信接口可能增加字段，验证签名时必须支持增加的扩展字段。

第二步，在 StringA 最后拼接上 key=（API 密钥的值）得到 stringSignTemp 字符串，并对 stringSignTemp 进行 MD5 运算，再将得到的字符串所有字符转换为大写，得到的值即最终的签名值。

由于在微信支付的过程中，所有的请求与响应都是需要生成签名或验证签名的，所以在代码实现的过程中要考虑到代码的复用性。下面将根据生成签名的步骤编写具体实现代码。

第一步，根据参数实体生成数据集合 M，并移除值为空的参数，如代码清单 7-1 所示。

代码清单 7-1

```
/// <summary>
/// 将实体对象转换成键值对。移除值为 null 的属性
/// </summary>
/// <param name="obj">参数实体</param>
/// <returns>数据键值对</returns>
```



```
public static Dictionary<string, string> EntityToDictionary(object obj)
{
    var type = obj.GetType();
    var dic = new Dictionary<string, string>();
    var pis = type.GetProperties();
    foreach (var pi in pis)
    {
        //获取属性的值
        var val = pi.GetValue(obj, null);
        //移除值为 null, 以及字符串类型的值为空字符的。另外, 签名字符串本身不参与签名,
        //在验证签名正确性时, 需移除 sign
        if (val == null || val.ToString() == "" || pi.Name == "sign") continue;
        dic.Add(pi.Name, val.ToString());
    }
    return dic;
}
```

第二步, 将参数集合内非空参数值的参数按照参数名 ASCII 码从小到大排序(字典序), 然后使用 URL 键值对的格式(即 key1=value1&key2=value2……)拼接成字符串 stringA, 拼接 key 后, MD5 加密, 并转换为大写。具体如代码清单 7-2 所示。

代码清单 7-2

```
///<summary>
///获取支付签名
///</summary>
///<param name="dictionary">数据集合</param>
///<param name="key">支付密钥</param>
///<returns>签名</returns>
public static string GetPaySign(Dictionary<string, string> dictionary,
    string key)
{
    var arr = dictionary.OrderBy(d => d.Key).Select(d => string.Format("{0}=",
        {1}", d.Key, d.Value)).ToArray();
    string stringA = string.Join("&", arr);
    return MD5(string.Format("{0}&key={1}", stringA, key)).ToUpper();
}
```

另外, GetPaySign 有个重载方法, 传入的参数分别是参数实体和支付密钥 key, 如代



码清单 7-3 所示。

代码清单 7-3

```
public static string GetPaySign(object obj, string key)
{
    var dic = EntityToDictionary(obj);
    return GetPaySign(dic, key);
}
```

以上所讲的都是生成签名的方法。为了保证交易安全，用户请求之后需验证返回信息里的签名是否是合法的。验证签名的实现代码如代码清单 7-4 所示。

代码清单 7-4

```
/// <summary>
/// 验证签名
/// </summary>
/// <param name="obj">参数集实体</param>
/// <param name="sign">签名</param>
/// <param name="key">支付密钥</param>
/// <returns></returns>
public static bool ValidSign(object obj, string sign, string key)
{
    var tempsign = GetPaySign(obj, key);
    return tempsign == sign;
}
```

微信支付 API 接口协议中包含字段 `nonce_str`，主要保证签名不可预测。笔者推荐的算法是生成 32 位的 GUID。因为 GUID 是不可预测且唯一的，并且长度也满足 `nonce_str` 的长度要求，如代码清单 7-5 所示。

代码清单 7-5

```
public static string GetGuid()
{
    return Guid.NewGuid().ToString("N");
}
```

在微信支付接口中，涉及资金回滚的接口会使用到商户证书，包括退款、撤销接口。



商家在申请微信支付成功后，收到相应的邮件，按照指引下载 API 证书。开发者在使用商户证书时，需确认 Framework 版本大于 2.0，且必须在操作系统上安装证书 `apiclient_cert.p12` 后才能被正常调用。商户证书调用或安装都需要使用到密码，该密码的值为微信商户号 (`mch_id`)。需要特别注意的是，证书文件不能放在 Web 服务器虚拟目录中，应放在有访问权限控制的目录中，以防被他人下载。

7.3 统一下单接口

除被扫支付场景以外，商户系统先调用该接口在微信支付服务后台生成预支付交易单，返回正确的预支付交易会话标识后再按扫码、JSAPI、APP 等不同场景生成交易串调起支付。接口地址如下：

`https://api.mch.weixin.qq.com/pay/unifiedorder`

请求参数如表 7-3 所示。

表 7-3

字段名	变量名	是否必填	类型	说明
公众账号 ID	<code>appid</code>	是	<code>string(32)</code>	微信分配的公众账号 ID
商户号	<code>mch_id</code>	是	<code>string(32)</code>	微信支付分配的商户号
设备号	<code>device_info</code>	否	<code>string(32)</code>	终端设备号（门店号或收银设备 ID），注意：PC 网页或公众号内支付请传“WEB”
随机字符串	<code>nonce_str</code>	是	<code>string(32)</code>	随机字符串，不长于 32 位
签名	<code>sign</code>	是	<code>string(32)</code>	签名，详见签名生成算法
商品描述	<code>body</code>	是	<code>string(32)</code>	商品或支付单简要描述
商品详情	<code>detail</code>	否	<code>string(8192)</code>	商品名称明细列表
附加数据	<code>attach</code>	否	<code>string(127)</code>	附加数据，在查询 API 和支付通知中原样返回，该字段主要用于商户携带订单的自定义数据
商户订单号	<code>out_trade_no</code>	是	<code>string(32)</code>	商户系统内部的订单号，32 个字符内、可包含字母
货币类型	<code>fee_type</code>	否	<code>string(16)</code>	符合 ISO 4217 标准的 3 位字母代码，默认人民币：CNY
总金额	<code>total_fee</code>	是	<code>int</code>	订单总金额，只能为整数，单位：分
终端 IP	<code>spbill_create_ip</code>	是	<code>string(16)</code>	App 和网页支付提交用户端 IP，Native 支付填调用微信支付 API 的机器 IP
交易起始时间	<code>time_start</code>	否	<code>string(14)</code>	订单生成时间，格式为 <code>yyyyMMddHHmmss</code>



续表

字段名	变量名	是否必填	类型	说明
交易结束时间	time_expire	否	string(14)	订单失效时间, 格式为 yyyyMMddHHmmss
商品标记	goods_tag	否	string(32)	商品标记, 代金券或立减优惠功能的参数
通知地址	notify_url	是	string(256)	接收微信支付异步通知回调地址
交易类型	trade_type	是	string(16)	取值如下: JSAPI, NATIVE, APP, WAP
商品 ID	product_id	否	string(32)	trade_type=NATIVE, 此参数必传。此 ID 为二维码中包含的商品 ID, 商户自行定义
用户标识	openid	否	string(128)	trade_type=JSAPI, 此参数必传, 是用户在商户 AppID 下的唯一标识。下单前需要调用“网页授权获取用户信息”接口获取到用户的 openid

根据表 7-3 中的字段创建实体类 UnifiedEntity, 此类继承自 BasePay, BasePay 的代码如代码清单 7-6 所示。

代码清单 7-6

```
namespace WxApi.PayEntity
{
    /// <summary>
    ///微信支付基类
    /// </summary>
    public abstract class BasePay
    {
        public string appid { get; set; }
        public string mch_id { get; set; }
        public string nonce_str { get; set; }
    }
}
```

UnifiedEntity 类的代码如代码清单 7-7 所示。

代码清单 7-7

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 统一支付接口请求实体
    /// </summary>
```



```
public class UnifiedEntity : BasePay
{
    /// <summary>
    /// 微信支付分配的终端设备号
    /// </summary>
    public string device_info { get; set; }
    /// <summary>
    /// 签名
    /// </summary>
    public string sign { get; set; }
    /// <summary>
    /// 商品描述
    /// </summary>
    public string body { get; set; }
    /// <summary>
    /// 商品详情
    /// </summary>
    public string detail { get; set; }
    /// <summary>
    /// 附加数据，原样返回
    /// </summary>
    public string attach { get; set; }
    /// <summary>
    /// 商户系统内部的订单号，32个字符内、可包含字母、确保在商户系统唯一
    /// </summary>
    public string out_trade_no { get; set; }
    /// <summary>
    /// 货币类型
    /// </summary>
    public string fee_type { get; set; }
    /// <summary>
    /// 订单总金额，单位为分，不能带小数点
    /// </summary>
    public int total_fee { get; set; }
    /// <summary>
    /// 终端 IP
    /// </summary>
    public string spbill_create_ip { get; set; }
```



```
    /// <summary>
    /// 交易起始时间
    /// </summary>
    public string time_start { get; set; }
    /// <summary>
    /// 交易结束时间
    /// </summary>
    public string time_expire { get; set; }
    /// <summary>
    /// 商品标记
    /// </summary>
    public string goods_tag { get; set; }
    /// <summary>
    /// 接收微信支付成功通知
    /// </summary>
    public string notify_url { get; set; }
    /// <summary>
    /// JSAPI、NATIVE、APP
    /// </summary>
    public string trade_type { get; set; }
    /// <summary>
    /// 用户在商户 AppID 下的唯一标识, trade_type 为 JSAPI 时, 此参数必传
    /// </summary>
    public string openid { get; set; }
    /// <summary>
    /// 只在 trade_type 为 NATIVE 时需要填写。此 ID 为二维码中包含的商品 ID, 商
    /// 户自行维护
    /// </summary>
    public string product_id { get; set; }
}
}
```

在调用统一下单接口前, 需要根据请求的参数生成签名, 然后再将参数列表生成代码清单 7-8 所示的 XML 数据, 并 POST 到接口。

代码清单 7-8

```
<xml>
  <return_code><![CDATA[SUCCESS]]></return_code>
```




```
<return_msg><![CDATA[OK]]></return_msg>
<appid><![CDATA[wx2421b1c4370ec43b]]></appid>
<mch_id><![CDATA[10000100]]></mch_id>
<nonce_str><![CDATA[IITRi8Iabbb1z1Jc]]></nonce_str>
<sign><![CDATA[7921E432F65EB8ED0CE9755F0E86D72F]]></sign>
<result_code><![CDATA[SUCCESS]]></result_code>
<prepay_id><![CDATA[wx201411101639507cbf6ffd8b0779950874]]></prepay_id>
<trade_type><![CDATA[JSAPI]]></trade_type>
</xml>
```

请求成功后，返回的示例代码如代码清单 7-9 所示。

代码清单 7-9

```
<xml>
  <return_code><![CDATA[SUCCESS]]></return_code>
  <return_msg><![CDATA[OK]]></return_msg>
  <appid><![CDATA[wx2421b1c4370ec43b]]></appid>
  <mch_id><![CDATA[10000100]]></mch_id>
  <nonce_str><![CDATA[IITRi8Iabbb1z1Jc]]></nonce_str>
  <sign><![CDATA[7921E432F65EB8ED0CE9755F0E86D72F]]></sign>
  <result_code><![CDATA[SUCCESS]]></result_code>
  <prepay_id><![CDATA[wx201411101639507cbf6ffd8b0779950874]]></prepay_id>
  <trade_type><![CDATA[JSAPI]]></trade_type>
</xml>
```

详细的参数列表如表 7-4 所示。

表 7-4

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看 result_code 来判断
返回信息	return_msg	string(128)	返回信息。如非空，为错误原因
以下字段在 return_code 为 SUCCESS 的时候有返回			
公众账号 ID	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
设备号	device_info	string(32)	终端设备号（门店号或收银设备 ID）。注意：PC 网页或公众号内支付请传“WEB”



续表

字段名	变量名	类型	说明
随机字符串	nonce_str	string(32)	随机字符串, 不长于 32 位
签名	sign	string(32)	签名, 详见签名生成算法
业务结果	result_code	string(16)	SUCCESS/FAIL
错误代码	err_code	string(32)	
错误代码描述	err_code_des	string(128)	错误返回的信息描述
以下字段在 return_code 和 result_code 都为 SUCCESS 的时候有返回			
交易类型	trade_type	string(16)	调用接口提交的交易类型, 取值如下: JSAPI, NATIVE, APP
预支付交易会 话标识	prepay_id	string(64)	微信生成的预支付会话标识, 用于在后续接口调用中使用, 该值有效期为 2 小时
二维码链接	code_url	string(64)	trade_type 为 NATIVE 时有返回, 可将该参数值生成二维码展示出来进行扫码支付

在上述参数列表中, 大部分参数和微信支付其他接口调用之后返回的参数是一样的, 只有部分参数是此接口专有的, 如 code_url、prepay_id 等。为了方便调用, 此处根据表 7-4 创建公共返回参数的基础类 BasePayRes, 如代码清单 7-10 所示。

代码清单 7-10

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 支付接口调用后, 响应参数基础类
    /// </summary>
    public class BasePayRes:BasePay
    {
        /// <summary>
        /// SUCCESS/FAIL, 此字段是通信标识, 非交易标识, 交易是否成功需要查看 result_
        /// code 来判断
        /// </summary>
        public string return_code { get; set; }
        /// <summary>
        /// 返回信息。如非空, 为错误原因
        /// </summary>
        public string return_msg { get; set; }
        /// <summary>
        /// 签名
    }
```



```
    /// </summary>
    public string sign { get; set; }
    /// <summary>
    /// 业务结果
    /// </summary>
    public string result_code { get; set; }
    /// <summary>
    /// 错误代码
    /// </summary>
    public string err_code { get; set; }
    /// <summary>
    /// 错误代码描述
    /// </summary>
    public string err_code_des { get; set; }
}
}
```

基类创建完毕后, 根据返回信息的参数列表创建子类 UnifiedRes, 如代码清单 7-11 所示。

代码清单 7-11

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 调用统一支付接口后, 返回的实体
    /// </summary>
    public class UnifiedRes : BasePayRes
    {
        /// <summary>
        /// 设备号
        /// </summary>
        public string device_info { get; set; }
        /// <summary>
        /// 预支付 ID
        /// </summary>
        public string prepay_id { get; set; }
        /// <summary>
        /// 交易类型
        /// </summary>
        public string trade_type { get; set; }
        /// <summary>
```



```
/// 二维码链接  
/// </summary>  
public string code_url { get; set; }  
}  
}
```

在请求接口时,需要将数据集合转换成 XML 数据包。在 Utils 类中添加如代码清单 7-12 所示的方法。

代码清单 7-12

```
public static string parseXML(Dictionary<string, string> parameters)  
{  
    StringBuilder sb = new StringBuilder();  
    sb.Append("<xml>");  
    var arr = parameters.Select(p => string.Format(Regex.IsMatch(p.Value,  
@"^[0-9.]+$") ? "<{0}>{1}</{0}>" : "<{0}><![CDATA[{1}]]></{0}>", p.Key,  
p.Value));  
    sb.Append(string.Join("", arr));  
    sb.Append("</xml>");  
    return sb.ToString();  
}
```

在请求成功后,需要将接收到的 XML 数据包转换给对应的实体类。实现代码如代码清单 7-13 所示。

代码清单 7-13

```
public static T XmlToEntity<T>(string xml)  
{  
    var type = typeof(T);  
    //创建实例  
    var t = Activator.CreateInstance<T>();  
    var pr = type.GetProperties();  
    var xdoc = XElement.Parse(xml);  
    foreach (var element in xdoc.Elements())  
    {  
        var p = pr.First(f => f.Name == element.Name);  
        p.SetValue(t, Convert.ChangeType(element.Value, p.PropertyType),  
null);  
    }  
}
```



```
return t;  
}
```

根据上述描述，微信支付的相关接口调用步骤总结起来就是：

- (1) 将请求实体转换成数据集合。
- (2) 生成签名，并将签名添加到数据集合。
- (3) 将数据集合转换成 XML。
- (4) 发送 POST 请求，并将返回信息转换为实体。

根据上述步骤新建类 `Pay`，用于封装微信支付相关的方法操作。然后在类中添加方法 `PayRequest`，方法的功能就是实现微信支付相关接口的调用，如代码清单 7-14 所示。

代码清单 7-14

```
/// <summary>  
/// 微信支付接口请求  
/// </summary>  
/// <typeparam name="T">返回值类型</typeparam>  
/// <param name="obj">请求实体</param>  
/// <param name="key">支付密钥</param>  
/// <param name="url">接口地址</param>  
/// <param name="certpath">证书路径，为 null 时说明不适用证书</param>  
/// <param name="certpwd">证书密码</param>  
public static T PayRequest<T>(object obj, string key, string url, string  
certpath="", string certpwd="")  
{  
    var param = Utils.EntityToDictionary(obj); // 将实体转换成数据集合  
    param.Add("sign", Utils.GetPaySign(param, key));  
    // 生成签名，并将签名添加到数据集合  
    var xml = Utils.parseXML(param); // 将数据集合转换成 XML  
    return Utils.XmlToEntity<T>(Utils.HttpPost(url, xml, certpath, certpwd));  
}
```

由于在微信支付的相关操作中，有的请求需要调用证书（退款、微信红包等），所以需要
要将 `Utils` 类中的 `HttpPost` 方法按照如图 7-2 所示的方式进行修改。



```
public static string HttpPost(string url, string data, string certpath="", string certpwd="")
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
    //当请求为https时, 验证服务器证书
    ServicePointManager.ServerCertificateValidationCallback = new RemoteCertificateValidationCallback((a, b, c, d) =>
    {
        if (d == SslPolicyErrors.None)
            return true;
        return false;
    });
    if (!string.IsNullOrEmpty(certpath) && !string.IsNullOrEmpty(certpwd))
    {
        X509Certificate2 cer = new X509Certificate2(certpath, certpwd,
            X509KeyStorageFlags.PersistKeySet | X509KeyStorageFlags.MachineKeySet);
        request.ClientCertificates.Add(cer);
    }
    request.Method = "POST";
    request.ContentType = "application/x-www-form-urlencoded";
    request.Accept = "*/*";
}
```

图 7-2

所以, 调用统一下单接口的代码如代码清单 7-15 所示。

代码清单 7-15

```
public static UnifiedRes UnifiedOrder(UnifiedEntity entity, string key)
{
    var url = "https://api.mch.weixin.qq.com/pay/unifiedorder";
    return PayRequest<UnifiedRes>(entity, key, url);
}
```

7.4 支付结果通用通知

在调用统一下单接口时, 参数 `notify_url` 表示的是接收支付结果通知的 URL。在支付完成后, 微信会把相关支付结果和用户信息发送给此 URL, 开发者需要接收并处理, 还要返回回答。与后台通知交互时, 如果微信收到商户服务器的应答不是成功或超时, 微信服务器就认为通知失败, 会通过一定的策略(如 30 分钟共 8 次)定期重新发起通知, 尽可能提高通知的成功率(但微信不保证通知最终能成功)。

由于存在重新发送后台通知的情况, 因此同样的通知可能会多次发送给商户系统。商户系统必须能够正确处理重复的通知。推荐的做法是, 当收到通知进行处理时, 首先检查对应业务数据的状态, 判断该通知是否已经处理过。如果没有处理过再进行处理; 如果处理过, 则返回成功结果。在对业务数据进行状态检查和处理之前, 要采用数据锁进行并发控制, 以避免函数重入造成的数据混乱。



通知的参数列表如表 7-5 所示。

表 7-5

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看 trade_state 来判断
返回信息	return_msg	string(128)	返回信息。如非空，为错误原因
以下字段在 return_code 为 SUCCESS 的时候有返回			
公众账号 ID	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
随机字符串	nonce_str	string(32)	
签名	sign	string(32)	
业务结果	result_code	string(16)	SUCCESS/FAIL
错误代码	err_code	string(32)	
错误代码描述	err_code_des	string(128)	
以下字段在 return_code 和 result_code 都为 SUCCESS 的时候有返回			
设备号	device_info	string(32)	微信支付分配的终端设备号
用户标识	openid	string(128)	用户在商户 AppID 下的唯一标识
是否关注公众账号	is_subscribe	string(1)	用户是否关注公众账号，Y——关注，N——未关注，仅在公众账号类型支付有效
交易类型	trade_type	string(16)	调用接口提交的交易类型，取值如下：JSAPI，NATIVE，APP，MICROPAY
付款银行	bank_type	string(16)	银行类型，采用字符串类型的银行标示
总金额	total_fee	int	订单总金额，单位为分
货币种类	fee_type	string(8)	默认人民币：CNY
现金支付金额	cash_fee	int	
现金支付货币类型	cash_fee_type	string(16)	
代金券或立减优惠金额	coupon_fee	int	“代金券或立减优惠”金额≤订单总金额，订单总金额-“代金券或立减优惠”金额=现金支付金额
代金券或立减优惠使用数量	coupon_count	int	
代金券或立减优惠批次 ID	coupon_batch_id_\$n	string(20)	代金券或立减优惠批次 ID，\$n 为下标，从 0 开始编号
代金券或立减优惠 ID	coupon_id_\$n	string(20)	代金券或立减优惠 ID，\$n 为下标，从 0 开始编号
单个代金券或立减优惠支付金额	coupon_fee_\$n	int	单个代金券或立减优惠支付金额，\$n 为下标，从 0 开始编号
微信支付订单号	transaction_id	string(32)	
商户订单号	out_trade_no	string(32)	商户系统内部的订单号，当没提供 transaction_id 时需要传这个
商家数据包	attach	string(128)	商家数据包，原样返回
支付完成时间	time_end	string(14)	订单支付时间，格式为 yyyyMMddHHmmss



请注意，在上述列表中“代金券或立减优惠批次 ID”、“代金券或立减优惠 ID”和“单个代金券或立减优惠支付金额”这 3 个字段的变量名是不定的，比如说当前订单使用了两个代金券，那么支付结果通知返回的 XML 中会包含如 coupon_fee_0、coupon_fee_1 类似的节点，这样的方式使我们调用反序列化方法很不便。因为返回的 XML 数据的节点是不定的，那么创建实体类也是一个问题。我的解决方法是将所有带有下标的字段封装为一个类，然后再在与 XML 相关联的实体类里添加一个泛型列表，在序列化与反序列化的时候进行特殊处理。

根据上述列表，创建实体类 OrderInfo，如代码清单 7-16 所示。

代码清单 7-16

```
namespace WxApi.PayEntity
{
    public class OrderInfo:BasePayRes
    {
        public string openid { get; set; }
        /// <summary>
        /// 是否订阅
        /// </summary>
        public string is_subscribe { get; set; }
        /// <summary>
        /// 交易类型
        /// </summary>
        public string trade_type { get; set; }
        /// <summary>
        /// 付款银行
        /// </summary>
        public string bank_type { get; set; }
        /// <summary>
        /// 总金额
        /// </summary>
        public int total_fee { get; set; }
        /// <summary>
        /// 代金券或立减优惠金额
        /// </summary>
        public string coupon_fee { get; set; }
        /// <summary>
```



```
/// 代金券或立减优惠使用数量
/// </summary>
public string coupon_count { get; set; }
/// <summary>
/// 货币种类
/// </summary>
public string fee_type { get; set; }
/// <summary>
/// 微信支付订单号
/// </summary>
public string transaction_id { get; set; }
/// <summary>
/// 商户订单号
/// </summary>
public string out_trade_no { get; set; }
/// <summary>
/// 商家数据包
/// </summary>
public string attach { get; set; }
/// <summary>
/// 支付完成时间
/// </summary>
public string time_end { get; set; }
/// <summary>
/// 现金支付金额
/// </summary>
public int cash_fee { get; set; }
public string cash_fee_type { get; set; }
public List<CouponInfo> CouponInfo { get; set; }
}

public class CouponInfo
{
    /// <summary>
    /// 代金券或立减优惠批次 ID
    /// </summary>
    public string coupon_batch_id { get; set; }
    /// <summary>
    /// 代金券或立减优惠 ID

```



```
/// </summary>
public string coupon_id { get; set; }
/// <summary>
/// 单个代金券或立减优惠支付金额
/// </summary>
public int coupon_fee { get; set; }
}
}
```

若将接收到的 XML 数据包反序列化为上述的实体类，则需要修改代码清单 7-12 中的方法。具体逻辑是，查找所有形如“_数字”的节点（如：coupon_fee_0、coupon_fee_1），判断查找到的结果。如果有，则将所有满足条件的节点转换为方便处理的匿名对象列表，并获取最大的数字。可根据此数据的大小判断实体中列表的项的数量。然后再转换为对应类型的列表。修改后的 XmlToEntity 方法如代码清单 7-17 所示。

代码清单 7-17

```
public static T XmlToEntity<T>(string xml)
{
    var type = typeof(T);
    //创建实例
    var t = Activator.CreateInstance<T>();
    var pr = type.GetProperties();
    var xdoc = XElement.Parse(xml);
    var eles = xdoc.Elements();
    var ele = eles.Where(e => new Regex(@"_d{1,}$").IsMatch(e.Name.
ToString()));//获取带下标的节点
    if (ele.Count() > 0)
    {
        var selele = ele.Select(e =>
        {
            var temp = e.Name.ToString().Split('_');
            var index = int.Parse(temp[temp.Length - 1]);
            return new { Index = index, Property = e.Name.ToString().
Replace("_" + index.ToString(), ""), Value = e.Value };
        }); //转换为方便操作的匿名对象
        var max = selele.Max(m => m.Index); //获取最大索引的值
        var infos = pr.FirstOrDefault(f => f.PropertyType.IsGenericType);
        //获取类型为泛型的属性
```




```

if (infos != null)
{
    var infotype = infos.PropertyType.GetGenericArguments().First();
    //获取泛型的真实类型
    Type listType = typeof(List<>).MakeGenericType(new[] { infotype });
    //创建泛型列表
    var datalist = Activator.CreateInstance(listType); //创建对象
    var infoprs = infotype.GetProperties();
    for (int j = 0; j <= max; j++)
    {
        var temp = Activator.CreateInstance(infotype);
        var list = selele.Where(s => s.Index == 0);
        foreach (var v in list)
        {
            var p = infoprs.FirstOrDefault(f => f.Name == v.Property);
            if (p == null) continue;
            p.SetValue(temp, v.Value, null);
        }
        listType.GetMethod("Add").Invoke((object)datalist, new[] { temp });
        //将对象添加到列表中
    }
    infos.SetValue(t, datalist, null); //最后给泛型属性赋值
}
ele.Remove(); //将有下标的节点从集合中移除
}
foreach (var element in eles)
{
    var p = pr.First(f => f.Name == element.Name);
    p.SetValue(t, Convert.ChangeType(element.Value, p.PropertyType),
null);
}
return t;
}

```

同时，在将接收到的 XML 反序列化后，需要验证签名，而验证签名时又需要还原真实接收到的数据集合。所以要处理诸如包含 coupon_fee_0、coupon_fee_1 属性的实体，需要修改代码清单 7-1 中的 EntityToDictionary，修改后如代码清单 7-18 所示。

代码清单 7-18

```

public static Dictionary<string, string> EntityToDictionary(object obj, int?

```




```
index = null)
{
    var type = obj.GetType();
    var dic = new Dictionary<string, string>();
    var pis = type.GetProperties();
    foreach (var pi in pis)
    {
        //获取属性的值
        var val = pi.GetValue(obj, null);
        //移除值为null, 以及字符串类型的值为空字符的。另外, 签名字符串本身不参与签名,
        //在验证签名正确性时, 需移除 sign
        if (val == null || val.ToString() == "" || pi.Name == "sign" ||
pi.PropertyType.IsGenericType) continue;
        if (index!=null)
        {
            dic.Add(pi.Name + "_" + index, val.ToString());
        }
        else
        {
            dic.Add(pi.Name, val.ToString());
        }
    }
    var classlist = pis.Where(p => p.PropertyType.IsGenericType);
    foreach (var info in classlist)
    {
        var val = info.GetValue(obj, null);
        if (val == null)
        {
            continue;
        }
        int count = (int)info.PropertyType.GetProperty("Count").GetValue
(val, null);
        if (count>0)
        {
            for (int i = 0; i < count; i++)
            {
                object ol = info.PropertyType.GetMethod("get_Item").Invoke
(val, new object[] { i });
                var tem = EntityToDictionary(ol, i); //递归调用
                foreach (var t in tem)
                {
                    dic.Add(t.Key, t.Value);
                }
            }
        }
    }
}
```



```
    }  
    }  
    }  
    return dic;  
}
```

商户服务器根据接收到的通知信息处理业务逻辑，处理后同步返回给微信的参数如表 7-6 所示。

表 7-6

字段名	变量名	是否必填	类型	示例值	说明
返回状态码	return_code	是	string(16)	SUCCESS	SUCCESS/FAIL SUCCESS 表示商户接收通知成功并校验成功
返回信息	return_msg	否	string(128)	OK	返回信息。如非空，为错误原因

在 Pay 类中添加方法 GetNotifyRes，用于将微信服务器发送过来的 XML 数据包转换成对应的实体，并根据处理结果调用商户的订单逻辑，如代码清单 7-19 所示。

代码清单 7-19

```
public static void GetNotifyRes(string key, Action<OrderInfo> callBack)  
{  
    try  
    {  
        var reqdata = Utils.GetRequestData();  
        var rev = Utils.XmlToEntity<OrderInfo>(reqdata);  
        if (rev.return_code != "SUCCESS")  
        { BackMessage("通讯错误"); return; }  
        if (rev.result_code != "SUCCESS")  
        { BackMessage("业务出错"); return; }  
        if (rev.sign == Utils.GetPaySign(rev, key))  
        {  
            //回调函数，业务逻辑处理，处理结束后返回信息给微信  
            callBack(rev);  
        }  
    }  
    catch (Exception e)  
    {  
    }  
}
```



```
BackMessage("回调函数处理错误");
```

在上述代码中，BackMessage 方法是用于返回处理信息给微信服务器的，如代码清单 7-20 所示。

代码清单 7-20

```
public static void BackMessage(string msg = "")
{
    Dictionary<string, string> dic = new Dictionary<string, string>();
    if (msg != "")
    {
        dic.Add("return_code", "FAIL");
        dic.Add("return_msg", msg);
        HttpContext.Current.Response.Write(Utils.parseXML(dic));
    }
    else
    {
        dic.Add("return_code", "SUCCESS");
        HttpContext.Current.Response.Write(Utils.parseXML(dic));
    }
}
```

处理方法创建完毕后就是实现 notify_url 的处理接口了。首先在 WxTest 项目中创建一般处理程序 paynotify.ashx，然后在程序的 ProcessRequest 方法中按照如图 7-3 所示的方式进行调用。

```
private string key = "支付密钥";
3 个引用
public void ProcessRequest(HttpContext context)
{
    WxApi.Pay.GetNotifyRev(key, e =>
    {
        //订单业务逻辑处理。需要先判断订单是否已经被处理，如已经被处理则直接回复处理结果。
        if (true) //订单处理成功
        {
            WxApi.Pay.BackMessage();
        }
        else
        {
            WxApi.Pay.BackMessage("错误原因");
        }
    });
}
```

图 7-3



7.5 查询订单接口

该接口提供所有微信支付订单的查询，商户可以通过该接口主动查询订单状态，完成下一步的业务逻辑。

需要调用查询接口的情况如下：

- 当商户后台、网络、服务器等出现异常时，商户系统最终未接收到支付通知。
- 调用支付接口后，返回系统错误或未知交易状态情况。
- 调用被扫支付 API，返回 USERPAYING 的状态。
- 调用关单或撤销接口 API 之前，需确认支付状态。

接口地址是：<https://api.mch.weixin.qq.com/pay/orderquery>。

请求的参数如表 7-7 所示。

表 7-7

字段名	变量名	是否必填	类型	说明
公众号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
微信订单号	transaction_id	否	string(32)	微信的订单号，优先使用
商户订单号	out_trade_no	否	string(32)	商户系统内部的订单号，当没提供 transaction_id 时需要传这个
随机字符串	nonce_str	是	string(32)	
签名	sign	是	string(32)	

请求的 XML 示例如代码清单 7-21 所示。

代码清单 7-21

```
<xml>
  <appid>wx2421b1c4370ec43b</appid>
  <mch_id>10000100</mch_id>
  <nonce_str>ec2316275641faa3aacf3cc599e8730f</nonce_str>
  <transaction_id>1008450740201411110005820873</transaction_id>
```




```
<sign>FDD167FAA73459FD921B144BAF4F4CA2</sign>
</xml>
```

根据上述示例代码创建实体类 OrderQuery，如代码清单 7-22 所示。

代码清单 7-22

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 订单查询实体
    /// </summary>
    public class OrderQuery:BasePay
    {
        /// <summary>
        /// 微信的订单号，优先使用
        /// </summary>
        public string transaction_id { get; set; }
        /// <summary>
        /// 商户系统内部的订单号。transaction_id、out_trade_no 二选一。如果同时
        /// 存在，则优先级如下：transaction_id > out_trade_no
        /// </summary>
        public string out_trade_no { get; set; }
        public string sign { get; set; }
    }
}
```

请求成功后返回的参数列表和支付结果通知接口的参数基本一致，此接口比支付结果通知接口多了两个参数，如表 7-8 所示。

表 7-8

字段名	变量名	类型	说明
交易状态	trade_state	string(32)	SUCCESS——支付成功 REFUND——转入退款 NOTPAY——未支付 CLOSED——已关闭 REVOKED——已撤销 USERPAYING——用户支付中 PAYERROR——支付失败（其他原因，如银行返回失败）
交易状态描述	trade_state_desc	string(256)	对当前查询订单状态的描述和下一步操作的指引



根据表 7-8 所示, 创建实体类 OrderQueryRes, 如代码清单 7-23 所示。

代码清单 7-23

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 查询订单状态, 返回结果实体
    /// </summary>
    public class OrderQueryRes : OrderInfo
    {
        public string trade_state { get; set; }
        /// <summary>
        /// 交易状态描述
        /// </summary>
        public string trade_state_desc { get; set; }
    }
}
```

最终, 订单查询的具体实现如代码清单 7-24 所示。

代码清单 7-24

```
public static OrderQueryRes QueryOrder(OrderQuery orderQuery, string key)
{
    var url = "https://api.mch.weixin.qq.com/pay/orderquery";
    return PayRequest<OrderQueryRes>(orderQuery, key, url);
}
```

7.6 JSAPI (网页内) 支付接口

7.6.1 场景交互细节

网页内支付指的是支付是从网页内发起的, 用户打开商户网页选购商品, 发起微信支付请求, 进入支付流程。商户服务器在收到用户订单信息后, 调用统一下单接口, 返回预支付 ID (prepayid), 然后通过 JavaScript 调用 JS 支付接口, 用户成功支付单击“完成”按钮后, 商户的前端会收到 JavaScript 的返回值, 开发者可根据返回值做进一步的操



作。另外，用户支付成功后，微信服务器会通过回调地址通知商户服务器支付的结果，开发者可根据回调时微信服务器传的数据判断支付的结果，并根据具体的业务逻辑进行操作，如图 7-4 所示。

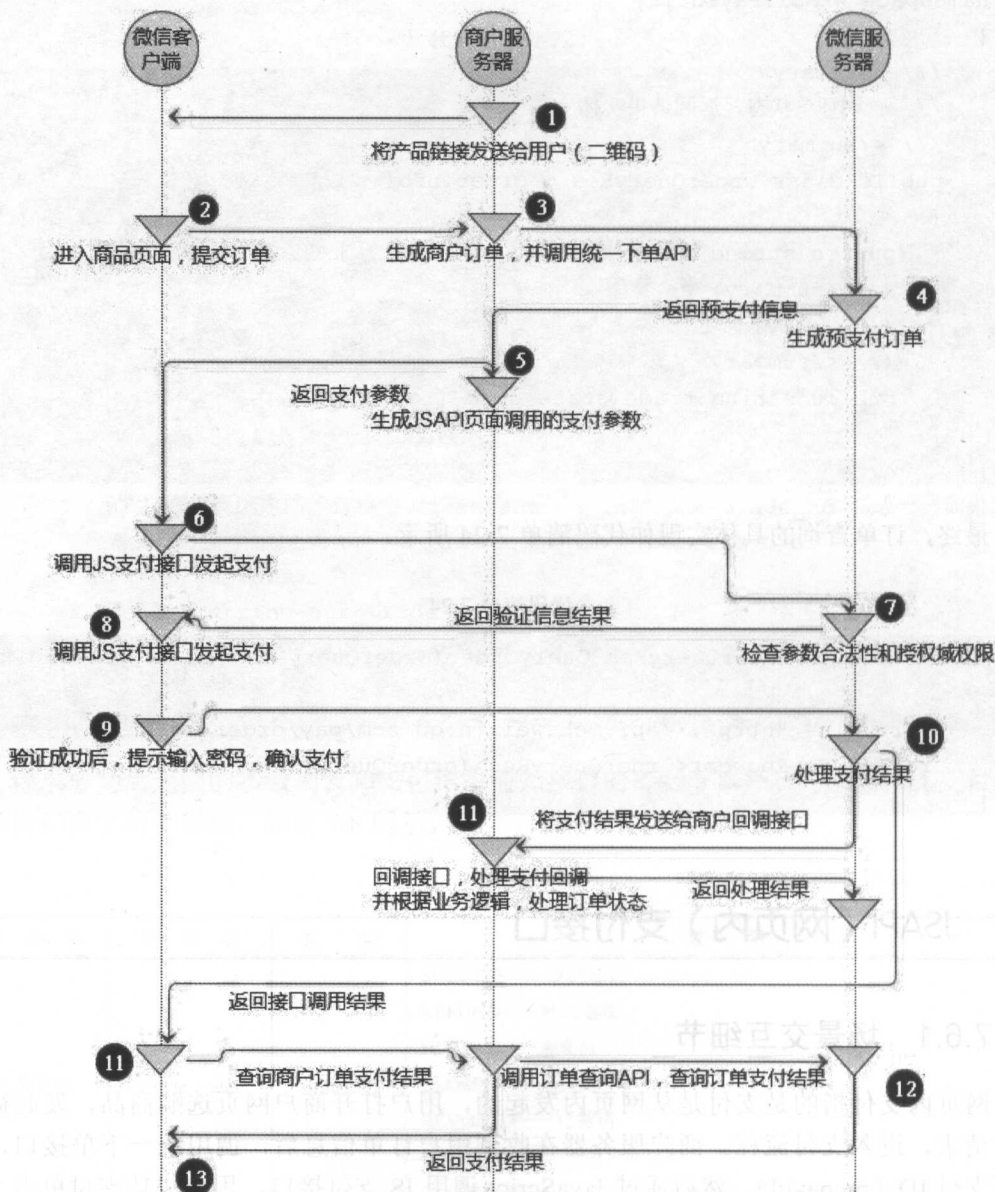


图 7-4



注意：微信的回调和前台 JS 的返回值并不保证遵循严格的时序，所以在图 7-4 所示的交互图中，可以看到两个“11”步骤。JS 返回值作为触发商户网页跳转的标志，商户后台应该只在收到微信后台的支付成功回调通知后，才做真正的支付成功的处理。或者当收到前台 JS 的返回值后，可调用订单查询接口查询订单的支付状态。如果已支付，则表明支付成功。

7.6.2 获取当前微信版本号

由于在微信 5.0 版本后才加入了微信支付模块，因此低版本用户调用微信支付功能将无效。故此，建议开发者通过 UA 来确定用户当前的版本号后再调用支付接口。以笔者的安卓手机为例，在微信浏览器中访问网页时的 UA 是：Mozilla/5.0 (Linux; U; Android 4.4.2; zh-cn; NX505J Build/KVT49L) AppleWebKit/533.1 (KHTML, like Gecko)Version/4.0 MQQBrower/5.4 TBS/025411 Mobile Safari/533.1 MicroMessenger/6.1.0.73_r1097298.543 NetType/WIFI。

从上述 UA 中可以看出，MicroMessenger/6.1 中的 6.1 就是微信的当前版本号。在实现的过程中，只需使用正则表达式解析出 UA 中和微信版本相关联的字符串即可，如代码清单 7-25 所示。

代码清单 7-25

```
public static decimal GetWxVersion()
{
    var ua = HttpContext.Current.Request.UserAgent;
    Regex reg = new Regex(@"MicroMessenger/(\d+).(\d+)");
    var result = reg.Match(ua).Groups;
    var temparr = result[0].Value.Split('/');
    return temparr.Length == 2 ? decimal.Parse(temparr[1]) : 0;
}
```

调用上述代码时，当返回值为 0 时，说明不是微信客户端，所以也可以使用上述方法检测是否是微信客户端打开的微网站。

7.6.3 显示微信安全支付标题

对于商户具有支付权限且需要调用微信支付的页面，为了让用户增加购买信息，确认交易环境安全，微信强烈建议商户使用“微信安全支付”标题。安全支付标题如图 7-5



所示。

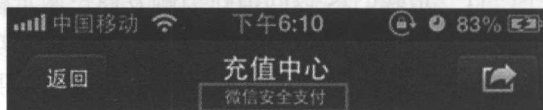


图 7-5

开发者只需将原始链接添加上“showwxpaytitle=1”的请求参数,通过这种方式,商户的页面将出现微信安全支付的标识。例如:原始支付 URL 为 <http://weixin.qq.com?a=123>,显示安全支付标题的 URL 应为 <http://weixin.qq.com?a=123&showwxpaytitle=1>。

7.6.4 JavaScript 调用支付 API

在微信浏览器中打开 H5 网页,调用 WeixinJSBridge 内置对象的 getBrandWCPayRequest 接口发起支付。接口输入/输出数据格式为 JSON。请注意:WeixinJSBridge 内置对象在其他浏览器中无效;列表中的参数名区分大小。

getBrandWCPayRequest 参数以及返回值定义如表 7-9 所示。

表 7-9

名 称	变 量 名	是 否 必 填	类 型	说 明
公众号 ID	appId	是	string(16)	
时间戳	timeStamp	是	string(32)	UNIX 时间戳
随机字符串	nonceStr	是	string(32)	
订单详情扩展字符串	package	是	string(128)	统一下单接口返回的 prepay_id 参数值,提交格式如: prepay_id=***
签名方式	signType	是	string(32)	签名算法,暂支持 MD5
签名	paySign	是	string(64)	

请求 JS 接口后, err_msg 的返回结果值如表 7-10 所示。

表 7-10

返 回 值	说 明
get_brand_wcpay_request:ok	支付成功
get_brand_wcpay_request:cancel	支付过程中用户取消
get_brand_wcpay_request:fail	支付失败



示例代码如代码清单 7-26 所示。

代码清单 7-26

```
function onBridgeReady(){
    WeixinJSBridge.invoke(
        'getBrandWCPayRequest', {
            "appId" : "wx2421b1c4370ec43b",    //公众号名称，由商户传入
            "timeStamp": "1395712654",        //时间戳，自 1970 年以来的秒数
            "nonceStr" : "e61463f8efa94090b1f366cccfbbb444", //随机串
            "package" : "prepay_id=u802345jgfjsdfgsdg888",
            "signType" : "MD5",                //微信签名方式
            "paySign" : "70EA570631E4BB79628FBCA90534C63FF7FADD89" //微信签名
        },
        function(res){
            if(res.err_msg == "get_brand_wcpay_request:ok" ) {}
            // 使用以上方式判断前端返回，微信团队郑重提示：res.err_msg 将在用户支付成
            // 功后返回 ok，但并不保证它绝对可靠
        }
    );
}

if (typeof WeixinJSBridge == "undefined"){
    if( document.addEventListener ){
        document.addEventListener('WeixinJSBridgeReady', onBridgeReady,
false);
    }else if (document.attachEvent){
        document.attachEvent('WeixinJSBridgeReady', onBridgeReady);
        document.attachEvent('onWeixinJSBridgeReady', onBridgeReady);
    }
}else{
    onBridgeReady();
}
```

7.6.5 网页内支付示例

1. 设置授权目录

设置授权目录包括测试目录与正式支付目录两种。开发者在开发测试的过程中，需要



设置测试目录，并将测试用户的微信号添加到白名单。发起支付的页面目录必须与设置的精确匹配，如支付的 URL 为 `http://www.abc.com/a/b/c.aspx`，正确的授权目录应该是 `http://www.abc.com/a/b/`，而 `http://www.abc.com/a/` 则是错误的授权目录。图 7-6 为设置测试目录的示意图。

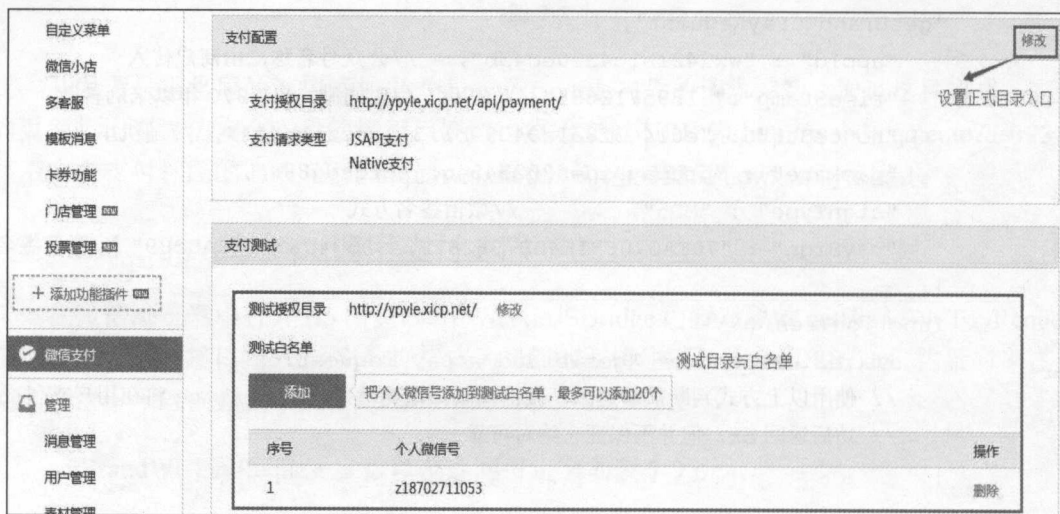


图 7-6

单击图 7-6 右上角的“修改”按钮，进入“微信支付设置”页面，可设置公众号支付的授权目录、Native 原生支付的回调 URL 以及告警通知 URL。支付的回调 URL 是用于接收在用户支付成功后，微信服务器推送的回调信息的。开发者需要根据相关的协议处理回调信息，并根据自己的业务逻辑处理订单状态。另外，告警通知 URL 是当商户服务器发生异常时，接收微信推送的消息的。由于微信官方并未提供相关的接口文档，且此 URL 并不会影响到真实的业务处理，因此开发者可以忽略。

2. 创建支付处理程序

在项目中创建一般处理程序 `WxPayHandler.ashx`，在处理程序中可配置支付相关的参数。根据请求的 `action` 参数判断当前的请求，如 `jspay` 说明当前请求的是网页内支付，`refund` 说明当前请求的是退款（见图 7-7）。

在处理程序中添加 `UnifiedOrder` 方法，用于获取请求的参数，并调用统一下单接口获取预支付 ID，如代码清单 7-27 所示。



```
public class WxPayHandler : IHttpHandler
{
    private const string key = "支付密钥";
    private const string notify_url = "http://{0}/paynotify.ashx";
    private const string appid = "appid";
    private const string mch_id = "商户号";
    3 个引用
    public void ProcessRequest(HttpContext context)
    {
        var action = context.Request.QueryString["action"];
        switch (action)
        {
            case "jspay": JsPay(context); break;
            case "refund": Refund(context); break;
            case "query": OrderIsSuccess(context); break;
        }
    }
}
```

图 7-7

代码清单 7-27

```
UnifiedRes UnifiedOrder(HttpContext context, string paytype)
{
    var param = new UnifiedEntity
    {
        appid = appid,
        mch_id = mch_id,
        nonce_str = Utils.GetGuid(),
        body = context.Request.Form["body"],
        detail = context.Request.Form["detail"],
        out_trade_no = context.Request.Form["out_trade_no"],
        spbill_create_ip = Utils.GetIP(),
        total_fee = int.Parse(context.Request.Form["total_fee"]),
        notify_url = string.Format(notify_url, Utils.GetCurrentFullHost()),
        trade_type = paytype,
        //openid 和 product_id 参数分别在 JSAPI 和 NATIVE 的支付类型中必填。调用的时
        //候根据类型给 msgid 赋值
        openid = context.Request.Form["msgid"],
        product_id = context.Request.Form["msgid"]
    };
    return WxApi.Pay.UnifiedOrder(param, key);
}
```



上述代码中的 `Utils.GetIP()` 方法的功能是获取当前服务器端的 IP，代码如下所示：

```
public static string GetIP()
{
    string result =
        HttpContext.Current.Request.ServerVariables["REMOTE_ADDR"];
    if (string.IsNullOrEmpty(result))
        result =
            HttpContext.Current.Request
                .ServerVariables["HTTP_X_FORWARDED_FOR"];
    if (string.IsNullOrEmpty(result))
        result = HttpContext.Current.Request.UserHostAddress;
    if (string.IsNullOrEmpty(result) ||
        !Regex.IsMatch(result,
            @"^((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)$"))
        return "127.0.0.1";
    return result;
}
```

根据代码清单 7-26 中的 JS 请求参数创建实体类 `JsEntity`，代码如下所示：

```
namespace WxApi.PayEntity
{
    public class JsEntity
    {
        /// <summary>
        /// 公众号 id
        /// </summary>
        public string appId { get; set; }
        /// <summary>
        /// 时间戳
        /// </summary>
        public string timeStamp { get; set; }
        /// <summary>
        /// 随机字符串
        /// </summary>
        public string nonceStr { get; set; }
        /// <summary>
        /// 订单详情扩展字符串
        /// </summary>
        public string package { get; set; }
        /// <summary>
        /// 签名类型
    }
}
```



```
/// </summary>
public string signType { get; set; }
/// <summary>
/// 签名
/// </summary>
public string paySign { get; set; }
}
}
```

添加 JsPay 方法，用户处理网页内支付的请求。此方法首先调用上述代码中的 UnifiedOrder 方法，根据统一下单接口返回的信息做进一步处理。假如返回的信息表示请求有误，则返回错误信息给客户端。如无误，则生成 JS 请求参数，并返回给客户端。具体代码如代码清单 7-28 所示。

代码清单 7-28

```
void JsPay(HttpContext context)
{
    var unifie = UnifiedOrder(context, "JSAPI");
    if (unifie.return_code == "SUCCESS" && Utils.ValidSign(unifie,
unifie.sign, key) && unifie.result_code == "SUCCESS")
    {
        var jsentity = new JsEntity
        {
            appId = appid,
            nonceStr = Utils.GetGuid(),
            package = "prepay_id=" + unifie.prepay_id,
            signType = "MD5",
            timeStamp = Utils.ConvertDateTimeInt(DateTime.Now).ToString()
        };
        jsentity.paySign = Utils.GetPaySign(jsentity, key);
        context.Response.Write(JsonConvert.SerializeObject(new { status =
"OK", info = jsentity }));
    }
    else if (unifie.return_code != "SUCCESS")
    {
        context.Response.Write(JsonConvert.SerializeObject(new { status =
"FAIL", info = unifie.return_msg }));
    }
    else if (unifie.result_code != "SUCCESS")
    {
        context.Response.Write(JsonConvert.SerializeObject(new { status =
"FAIL", info = unifie.err_code_des }));
    }
}
```




```
}  
else  
{  
    context.Response.Write(JsonConvert.SerializeObject(new { status =  
        "FAIL", info = "微信服务器返回的签名不合法!" }));  
}  
}
```

3. 客户端调用

新建页面 Pay.aspx, 在页面中添加如图 7-8 所示的表单。

商品名:	<input type="text" value="测试商品"/>
商品详情:	<input type="text"/>
订单号:	<input type="text" value="18095f18a5be4f65b1741bf"/>
订单金额:	<input type="text" value="1"/>
<input type="button" value="支付"/>	

图 7-8

当单击“支付”按钮时, 发送 POST 请求到 WxPayHandler.ashx, 获取 JS 支付参数后, 调用 7.6.4 节中所讲的 JS 方法即可发起支付, 如代码清单 7-29 所示。

代码清单 7-29

```
var time;  
var timecount=0;  
function onBridgeReady() {  
    $("#btnpay").click(function () {  
        layer.open({type:2,content:"正在创建订单...",shadeClose:false});  
        //加载 Loading 层  
        $.post("/WxPayHandler.ashx?action=jspay", $("form").serialize(),  
function (data) {  
    layer.closeAll();//关闭 Loading 层  
    if (data.status=="OK") {  
        var param = data.info;  
        WeixinJSBridge.invoke(  
            'getBrandWCPayRequest',  
            param,  
            function (res) {  
                if (res.err_msg == "get_brand_wcpay_request:ok") {
```




```
    } //使用以上方式判断前端返回。微信团队郑重提示: res.err_msg 将在用户
    //支付成功后返回 ok, 但并不保证它绝对可靠
    layer.open({type:2,content:"正在查询订单状态...",shadeClose:
false});

    time= setInterval(function(){
        queryorder($("#input[name='out_trade_no']").val()); }, 2000);
    })
    } else {
        alert(data.info);
    }
    }, "json");
});
}

if (typeof WeixinJSBridge == "undefined") {
    if (document.addEventListener) {
        document.addEventListener('WeixinJSBridgeReady', onBridgeReady,
false);
    } else if (document.attachEvent) {
        document.attachEvent('WeixinJSBridgeReady', onBridgeReady);
        document.attachEvent('onWeixinJSBridgeReady', onBridgeReady);
    }
} else {
    onBridgeReady();
}

//查询订单是否支付成功
function queryorder(tradeno) {
    if (timecount++<3) {
        $.post("/WxPayHandler.ashx?action=query", {out_trade_no:tradeno},
function(data,status) {
            if (data==1) {
                layer.closeAll();
                alert("支付成功!");
                clearInterval(time);
            }
        });
    } else {
        layer.closeAll();
        alert("订单貌似没有支付成功!");
        clearInterval(time);
    }
}
```



在上述代码中，由于调用一般处理程序有的时候会有延时，因此在调用之前加了一个 Loading 层（layer 弹出层插件），调用成功后关闭 Loading 层。在支付成功后，微信服务器会调用通知页面 notify_url，同时客户端 JS 会返回成功信息。但前端 JS 的返回值并不能保证可靠，此时需要调用订单查询接口查询订单的支付状态。如果状态为成功，则说明此笔订单支付成功。

在处理程序中，查询订单是否支付的代码如代码清单 7-30 所示。

代码清单 7-30

```
void OrderIsSuccess(HttpContext context)
{
    var query = new OrderQuery
    {
        appid = appid,
        mch_id = mch_id,
        nonce_str = Utils.GetGuid(),
        out_trade_no = context.Request.Form["out_trade_no"],
        transaction_id = context.Request.Form["transaction_id"]
    };
    var info = WxApi.Pay.QueryOrder(query, key);
    context.Response.Write(info.trade_state);
}
```

除了使用上述方法发起支付外，还可以使用 JS-SDK 提供的 chooseWXPAY 方法。调用方式如代码清单 7-31 所示。

代码清单 7-31

```
var param = data.info; //提交表单返回的参数
wx.chooseWXPAY({
    timestamp: param.timestamp, //支付签名时间戳，注意微信 JS-SDK 中使用的所有
    //timestamp 字段均为小写。但最新版的支付后台生成签名使用的 timestamp 字段名需大
    //写其中的 S 字符
    nonceStr: param.nonceStr, //支付签名随机串，不长于 32 位
    package: param.package,
    //统一支付接口返回的 prepay_id 参数值，提交格式如：prepay_id=***
    signType: param.signType, //签名方式，默认为 'SHA1'，使用新版支付需传入 'MD5'
    paySign: param.paySign, //支付签名
```



```
success: function (res) {  
    //支付成功后的回调函数  
},  
cancel:function(res) {  
    alert(JSON.stringify(res));  
}  
});
```

JS-SDK 使用方式请参考第 6 章，在此不再赘述。

7.7 扫码支付

从直观上理解，扫码支付就是扫描二维码发起支付。实现流程大致可以描述为：商户根据微信支付的规则，为不同商品生成不同的二维码，展示在各种场景，用于用户扫描购买。用户使用微信“扫一扫”扫描二维码后，获取商品支付信息，引导用户完成支付。

扫码支付可分为两种模式，商户可根据支付场景选择相应的模式。

模式一：商户后台系统根据微信支付规则链接生成二维码，用户扫码后，微信支付系统将 `productid` 和用户 `openid` 回调商户后台系统，商户后台系统根据系统 `productid` 生成支付交易，最后微信支付系统发起用户支付流程。

模式二：商户后台系统调用微信统一下单 API 生成预支付订单，将接口返回的链接生成二维码，用户扫描后输入密码完成支付交易。注意：该模式的预支付订单有效期为 2 小时，过期后无法支付。

7.7.1 扫描支付——模式一

模式一开发前，开发者必须在公众平台后台依次进入“微信支付”→“开发配置”→“支付配置”设置支付回调 URL，如图 7-9 所示。

URL 实现的功能是：接收用户扫码后微信支付系统回调的 `productid` 和 `openid`，根据相关协议生成订单并将订单信息回复给微信支付系统，微信支付系统再响应用户客户端的请求，进入支付流程。具体的业务流程如图 7-10 所示。



☒ Native原生支付 以线下扫码支付为代表的快速支付方式，详见支付接口文档

支付回调URL

头部要包括http或https，当公众平台接收到Native原生支付请求时，修改会影响线上交易，距正式生效有十分钟左右延迟，建议你避开

告警通知URL

头部要包括http或https，微信监测到商户服务出现问题时，会及时推送相关告警信息

图 7-9

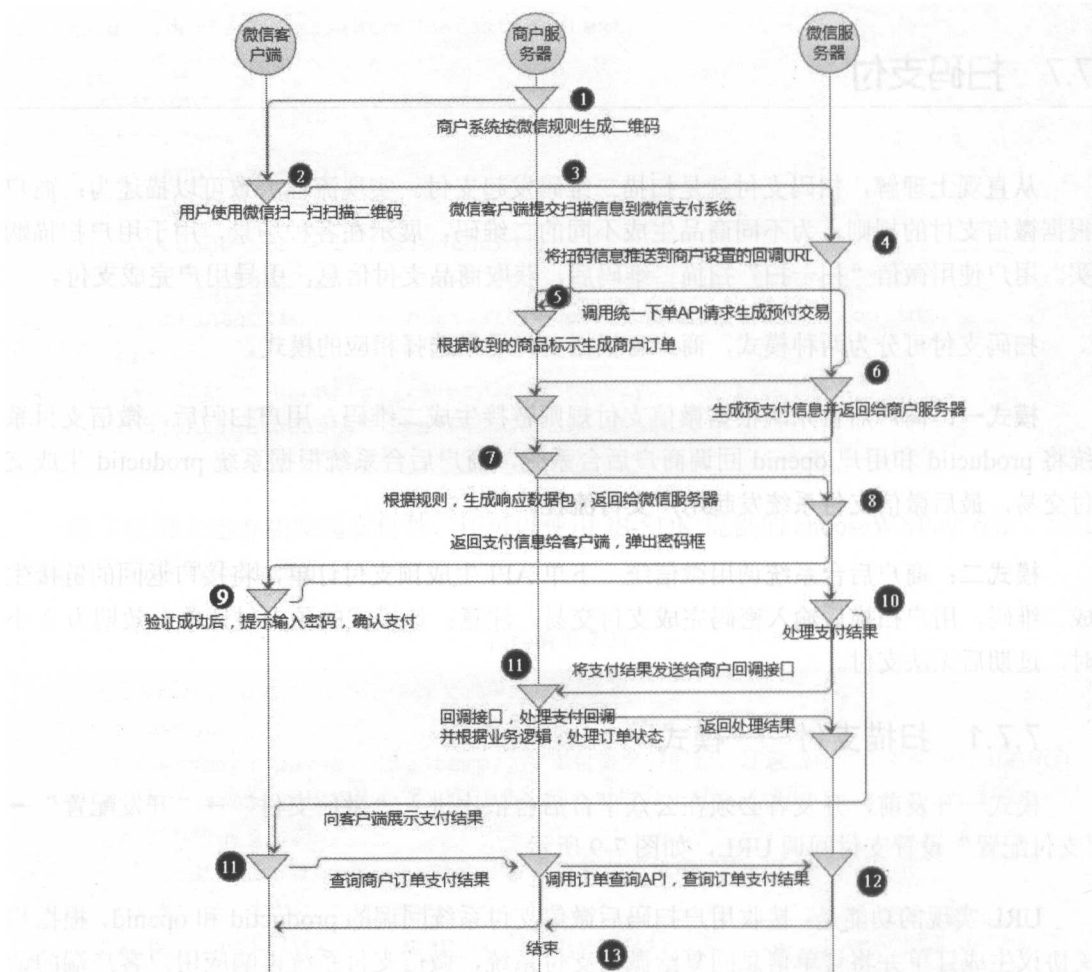


图 7-10



下面将根据图 7-10 的流程，一步步地实现模式一的支付流程。

1. 生成二维码

二维码中的内容为链接，形式如下：

```
weixin://wxpay/bizpayurl?sign=XXXXX&appid=XXXXX&mch_id=XXXXX&product_id=XXXXXX&time_stamp=XXXXXX&nonce_str=XXXXX
```

具体的参数如表 7-11 所示。

表 7-11

名 称	变 量 名	类 型	是 否 必 填	说 明
公众账号 ID	appid	string(32)	是	微信分配的公众账号 ID
商户号	mch_id	string(32)	是	
时间戳	time_stamp	string(32)	是	
随机字符串	nonce_str	string(32)	是	
商品 ID	product_id	string(32)	是	商户定义的商品 ID 或者订单号
签名	sign	string(32)	是	

根据表 7-11 中的参数创建实体类 NativeEntity，如代码清单 7-32 所示。

代码清单 7-32

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// native 支付二维码生成参数
    /// </summary>
    public class NativeEntity:BasePay
    {
        /// <summary>
        /// 时间戳
        /// </summary>
        public string time_stamp { get; set; }
        /// <summary>
        /// 商品 ID
        /// </summary>
        public string product_id { get; set; }
    }
}
```




```
    }  
}
```

在 Pay 类中添加生成二维码的链接，如代码清单 7-33 所示。

代码清单 7-33

```
public static string CreateNativeUrl (NativeEntity nativeEntity, string key)  
{  
    var param = Utils.EntityToDictionary(nativeEntity); //将实体转换成数据集合  
    param.Add("sign", Utils.GetPaySign(param, key));  
    //生成签名，并将签名添加到数据集合  
    var arr = param.Select(p => p.Key + "=" + p.Value);  
    return string.Format("weixin://wmpay/bizpayurl?{0}", string.Join("&", arr));  
}
```

2. 处理二维码扫描回调

用户扫描了上一步生成的二维码后，微信客户端会将二维码的信息推送到微信支付系统的服务器，然后微信支付系统服务器将信息推送到商户设置的 NATIVE 回调 URL。参数如表 7-12 所示。

表 7-12

名 称	变 量 名	类 型	是 否 必 填	说 明
公众账号 ID	appid	string(32)	是	微信分配的公众账号 ID
用户标识	openid	string(128)	是	
商户号	mch_id	string(32)	是	
是否关注	is_subscribe	string(1)	是	用户是否关注公众账号：Y——关注；N——未关注
随机字符串	nonce_str	string(32)	是	
商品 ID	product_id	string(32)	是	商户定义的商品 ID 或者订单号
签名	sign	string(32)	是	

根据表 7-12 中的参数创建实体类 NativeNotify，如代码清单 7-34 所示。

代码清单 7-34

```
namespace WxApi.PayEntity  
{  
    /// <summary>
```



```
/// 模式一扫描二维码回调
/// </summary>
public class NativeNotify:BasePay
{
    /// <summary>
    /// 用户在商户 AppID 下的唯一标识
    /// </summary>
    public string openid { get; set; }
    /// <summary>
    /// 是否关注公众号
    /// </summary>
    public string is_subscribe { get; set; }
    /// <summary>
    /// 商品 ID
    /// </summary>
    public string product_id { get; set; }
    /// <summary>
    /// 签名
    /// </summary>
    public string sign { get; set; }
}
```

接收到回调，调用统一下单接口获取预支付 ID 后，将订单信息返回给微信支付系统服务器，最后微信支付系统服务器再响应用户的请求。需要返回的参数列表如表 7-13 所示。

表 7-13

名 称	变 量 名	类 型	是 否 必 填	说 明
返回状态码	return_code	string(16)	是	SUCCESS/FAIL，此字段是通信标识，非交易标识，交易是否成功需要查看 result_code 来判断
返回信息	return_msg	string(128)	否	返回信息，如非空，为错误原因；签名失败；具体某个参数格式校验错误
公众账号 ID	appid	string(32)	是	微信分配的公众账号 ID
商户号	mch_id	string(32)	是	
随机字符串	nonce_str	string(32)	是	
预支付 ID	prepay_id	string(64)	是	调用统一下单接口生成的预支付 ID
业务结果	result_code	string(16)	是	SUCCESS/FAIL
错误描述	err_code_des	string(128)	否	当 result_code 为 FAIL 时，商户展示给用户的错误提示
签名	sign	string(32)	是	



根据表 7-13 的参数列表创建实体类 NativeRes，如代码清单 7-35 所示。

代码清单 7-35

```
namespace WxApi.PayEntity
{
    public class NativeRes:BasePayRes
    {
        public string prepay_id { get; set; }
    }
}
```

各个阶段的实体类创建完毕后，下面就是具体业务逻辑的实现了。首先在 Pay 类中添加方法 GetNotiveRes，在方法体中，首先获取当前请求的数据，并将数据转换成对应的实体类，然后调用 Func 委托，Func 委托是提供给回调页面调用生成订单、调用统一下单接口的方法。调用完成后，根据 Func 委托返回的实体生成响应微信支付系统服务器的实体，并转换成 XML 返回给微信支付系统服务器，如代码清单 7-36 所示。

代码清单 7-36

```
public static void GetNotiveRes(string key,Func<NativeNotify,UnifiedRes>
func)
{
    var reqdata = Utils.GetRequestData();
    var rev = Utils.XmlToEntity<NativeNotify>(reqdata);
    var unifie = func(rev); //获取委托返回的对象
    var res = new NativeRes
    {
        appid = unifie.appid,
        err_code_des = unifie.err_code_des,
        mch_id = unifie.mch_id,
        nonce_str = unifie.nonce_str,
        prepay_id = unifie.prepay_id,
        return_code = unifie.return_code,
        return_msg = unifie.return_msg,
        result_code = unifie.result_code
    };
    var param = Utils.EntityToDictionary(res); //将实体转换成数据集合
    param.Add("sign", Utils.GetPaySign(param, key));
    //生成签名，并将签名添加到数据集合
    var xml = Utils.parseXML(param); //将数据集合转换成 xml
```



```
HttpContext.Current.Response.Write(xml);
```

```
}
```

然后在 WxTest 项目中创建一般处理程序 `nativenotify`，用于处理 NATIVE 模式一扫描二维码后的回调。在页面的 `ProcessRequest` 方法中调用上述方法，创建订单后，再调用统一下单接口，如代码清单 7-37 所示。

代码清单 7-37

```
private string key = "支付密钥";
string notify_url = "统一下单 api 回调 url";
public void ProcessRequest(HttpContext context)
{
    WxApi.Pay.GetNotiveRes(key, e =>
    {
        //创建订单
        //开发者需要根据自己的业务逻辑创建订单。并把订单号作为统一下单 api 的商户订单号
        //创建订单
        var un = new UnifiedEntity
        {
            appid = e.appid,
            body = "测试商品",
            mch_id = e.mch_id,
            nonce_str = e.nonce_str,
            notify_url = string.Format(notify_url,
                Utils.GetCurrentFullHost()),
            out_trade_no = Utils.GetGuid(),
            openid = e.openid,
            product_id = e.product_id,
            spbill_create_ip = Utils.GetIP(),
            total_fee = 1,
            trade_type = "NATIVE"
        };
        return WxApi.Pay.UnifiedOrder(un, key); //调用统一下单 api
    });
}
```

至此，模式一的代码实现已全部完成。用户支付完成后，会回调调用统一下单 API 时设置的回调 URL。读者可参考 7.4 节的相关描述。

由于模式一的订单是在用户扫描后才生成的，因此此种模式适合那种只需要一个二维



码就可以让多个人购买的场景，比如自动售卖机。

7.7.2 扫描支付——模式二

模式二与模式一相比，流程更为简单，其不依赖设置的回调支付 URL。商户后台系统先调用微信支付的统一下单接口，微信支付系统会返回链接参数 `code_url`，商户后台系统将 `code_url` 的值生成二维码图片，使用微信客户端扫码后发起支付。`code_url` 有效期为 2 小时，过期扫码不能再发起支付。另外，此模式的一个二维码只能被一个用户支付。所以此模式适合 PC 版的微信支付或外卖场景下，客户扫码小票的二维码支付。如图 7-11 所示为调用统一下单接口返回的 `code_url`。

表达式(E):	responseStr
值(V):	<pre><![CDATA[1240050702]]> </mch_id> - <nonce_str> <![CDATA[ahyh6IFJvS05cCjh]]> </nonce_str> - <sign> <![CDATA[A557DB54B7AC827D23348790A81930C6]]> </sign> - <result_code> <![CDATA[SUCCESS]]> </result_code> - <prepay_id> <![CDATA[wx20150511153204f43f8cc0cf0728449441]]> </prepay_id> - <trade_type> <![CDATA[NATIVE]]> </trade_type> - <code_url> <![CDATA[weixin://wxpay/bizpayurl?pr=20Zn7gP]]> </code_url> </xml></pre>

7-11

统一下单接口的调用方式请参考 7.4 节，生成二维码的方式也已经在 3.7.1 节中讲述了，在此不再赘述。

7.8 刷卡支付

刷卡支付的使用场景是：收银员使用扫码设备读取微信用户刷卡授权码以后，二维码



或条码信息传送至商户收银台，由商户收银台或者商户后台调用被扫支付接口发起支付。用户的授权码是从微信客户端“我”→“钱包”→“刷卡”中获取到的，如图 7-12 所示。



图 7-12

整个流程大致可以总结为如下几点：

- (1) 收银员在商户收银台生成支付订单，向用户展示支付金额。
- (2) 用户打开微信客户端，单击“我的钱包”，选择“刷卡”，进入条码界面。
- (3) 使用扫码设备读取用户手机屏幕上的条码。
- (4) 扫码设备将读取的信息上传给门店收银台。
- (5) 门店收银台得到支付信息后，向商户收银后台发起支付请求。
- (6) 商户后台对门店收银台的支付请求进行处理，生成签名后调用提交被扫支付 API



向微信支付系统发起支付请求。

(7) 微信支付系统得到商户侧的支付请求之后会对请求进行验证, 验证通过之后会对请求数据进行处理, 最后将处理后的支付结果返回给商户收银后台。如果支付成功, 则微信支付系统会将支付结果返回给商户, 同时把支付结果通知给用户 (以短信、微信消息的形式通知)。

(8) 商户收银后台对得到的支付结果进行签名验证和处理, 再将支付结果返回给门店收银台。

(9) 收银员看到门店收银台的支付结果后给用户发货。

需要注意的是, 刷卡支付分为免密模式和有密模式。对于支付金额大于 1000 元的交易, 需要验证密码。用户账户每天最多有 5 笔交易可以免密; 超过 5 笔交易的以及微信支付后台判断用户支付行为有异常情况时, 即使符合免密规则的交易也会要求验证密码。假如交易需要验证密码, 则在商户后台调用被扫支付 API 后, 返回的 XML 信息中的 `err_code` 的值为 `USERPAYING`, 如图 7-13 所示。

```
<xml><return_code><![CDATA[SUCCESS]]></return_code>
<return_msg><![CDATA[OK]]></return_msg>
<appid><![CDATA[wx891ee9903ba8d74e]]></appid>
<mch_id><![CDATA[1240050702]]></mch_id>
<nonce_str><![CDATA[1MMF9PXDCwtWiFzz]]></nonce_str>
<sign><![CDATA[BCA366FE93E30BF7B7375C395EC48F53]]></sign>
<result_code><![CDATA[FAIL]]></result_code>
<err_code><![CDATA[USERPAYING]]></err_code>
<err_code_des><![CDATA[需要用户输入支付密码]]></err_code_des>
</xml>
```

图 7-13

在这种情况下, 微信支付系统需要通知用户微信客户端输入密码, 用户得到输入密码提示后, 确认支付并输入密码。商户系统收到 `USERPAYING` 状态后, 商户收银系统每隔 5 秒循环调用“查询订单 API”查询实际支付结果。如果用户取消支付或累计 30 秒用户都未支付, 则商户系统停止循环查询, 并调用“撤销订单 API”撤销支付交易。

在支付的过程中, 用户微信端弹出支付失败提示, 例如: 余额不足、信用卡失效等, 需要重新发起支付。当交易超时或支付失败时, 商户收银系统必须调用“撤销订单 API”撤销此交易。



被扫支付的接口地址是：<https://api.mch.weixin.qq.com/pay/micropay>。

发起请求时需要提交的参数如表 7-14 所示。

表 7-14

字段名	变量名	是否必填	类型	说明
公众账号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
设备号	device_info	否	string(32)	终端设备号（门店号或收银设备 ID），注意：PC 网页或公众号内支付请传“WEB”
随机字符串	nonce_str	是	string(32)	随机字符串，不长于 32 位
签名	sign	是	string(32)	签名，详见签名生成算法
商品描述	body	是	string(32)	商品或支付单简要描述
商品详情	detail	否	string(8192)	商品名称明细列表
附加数据	attach	否	string(127)	附加数据，在查询 API 和支付通知中原样返回，该字段主要用于商户携带订单的自定义数据
商户订单号	out_trade_no	是	string(32)	商户系统内部的订单号，32 个字符内、可包含字母
货币类型	fee_type	否	string(16)	符合 ISO 4217 标准的 3 位字母代码，默认为人民币：CNY
总金额	total_fee	是	int	订单总金额，只能为整数，单位：分
终端 IP	spbill_create_ip	是	string(16)	
交易起始时间	time_start	否	string(14)	订单生成时间，格式为 yyyyMMddHHmmss
交易结束时间	time_expire	否	string(14)	订单失效时间，格式为 yyyyMMddHHmmss
商品标记	goods_tag	否	string(32)	商品标记，代金券或立减优惠功能的参数
授权码	auth_code	是	string(128)	扫码支付授权码，设备读取用户微信中的条码或者二维码信息

根据表 7-14 创建实体类 ScanPayEntity，如代码清单 7-38 所示。

代码清单 7-38

```
namespace WxApi.PayEntity
{
    public class ScanPayEntity:BasePay
    {
        public string device_info { get; set; }
        public string sign { get; set; }
        public string body { get; set; }
        public string detail { get; set; }
```



```
public string attach { get; set; }
public string out_trade_no { get; set; }
public string total_fee { get; set; }
public string fee_type { get; set; }
public string spbill_create_ip { get; set; }
public string time_start { get; set; }
public string time_expire { get; set; }
public string goods_tag { get; set; }
public string auth_code { get; set; }
}
}
```

请求接口后,返回的信息和统一下单接口的返回信息是一致的。最后,在 Pay 类中添加调用此接口的方法,如代码清单 7-39 所示。

代码清单 7-39

```
public static OrderInfo ScanPay(ScanPayEntity scan, string key)
{
    var url = "https://api.mch.weixin.qq.com/pay/micropay";
    return PayRequest<OrderInfo>(scan, key, url );
}
```

7.9 撤销订单

在支付交易返回失败或支付系统超时,可调用撤销订单接口撤销交易。如果此订单用户支付失败,则微信支付系统会将此订单关闭。如果用户支付成功,则微信支付系统会将此订单资金退还给用户。

注意: 7 天以内的交易单可调用撤销,其他正常支付的单如需相同功能,则请调用申请退款 API。

接口地址是 <https://api.mch.weixin.qq.com/secapi/pay/reverse>,调用此接口需要双向证书。调用此接口请求的参数和查询订单接口的参数是一样的,在此不再赘述。调用接口返回的参数列表如表 7-15 所示。



表 7-15

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识, 非交易标识, 交易是否成功需要查看 result_code 来判断
返回信息	return_msg	string(128)	返回信息。如非空, 为错误原因
以下字段在 return_code 为 SUCCESS 的时候有返回			
公众账号 ID *	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
随机字符串	nonce_str	string(32)	随机字符串, 不长于 32 位
签名	sign	string(32)	签名, 详见签名生成算法
业务结果	result_code	string(16)	SUCCESS/FAIL
错误代码	err_code	string(32)	
错误代码描述	err_code_des	string(128)	错误返回的信息描述
是否重调	recall	string(1)	是否需要继续调用撤销: Y——需要, N——不需要

根据表 7-15 创建实体类 ReverseRes, 如代码清单 7-40 所示。

代码清单 7-40

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 撤销接口返回实体
    /// </summary>
    public class ReverseRes : BasePayRes
    {
        /// <summary>
        /// 是否需要继续调用撤销: Y——需要, N——不需要
        /// </summary>
        public string recall { get; set; }
    }
}
```

实体创建完毕后, 在 Pay 类中添加调用接口的方法 ReverseRes, 如代码清单 7-41 所示。

代码清单 7-41

```
public static ReverseRes ReverseOrder(OrderQuery order, string key, string
```




```
certpath, string certpwd)
{
    var url = "https://api.mch.weixin.qq.com/secapi/pay/reverse";
    return PayRequest<ReverseRes>(order, key, url, certpath, certpwd);
}
```

7.10 关闭订单

商户订单支付失败需要生成新单号重新发起支付，要对原订单号调用关单，避免重复支付；系统下单后，用户支付超时，系统退出不再受理，为避免用户继续，请调用关单接口。接口地址是：<https://api.mch.weixin.qq.com/pay/closeorder>。

调用接口请求的参数列表如表 7-16 所示。

表 7-16

字段名	变量名	是否必填	类型	说明
公众号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
商户订单号	out_trade_no	否	string(32)	商户系统内部的订单号
随机字符串	nonce_str	是	string(32)	
签名	sign	是	string(32)	

根据表 7-16 创建实体类 CloseOrder，如代码清单 7-42 所示。

代码清单 7-42

```
namespace WxApi.PayEntity
{
    public class CloseOrder : BasePay
    {
        /// <summary>
        /// 商户系统内部的订单号
        /// </summary>
        public string out_trade_no { get; set; }
        public string sign { get; set; }
    }
}
```



调用接口返回的参数列表如表 7-17 所示。

表 7-17

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看 result_code 来判断
返回信息	return_msg	string (128)	返回信息。如非空，为错误原因
以下字段在 return_code 为 SUCCESS 的时候有返回			
公众账号 ID	appid	string (32)	微信分配的公众账号 ID
商户号	mch_id	string (32)	微信支付分配的商户号
随机字符串	nonce_str	string (32)	随机字符串，不长于 32 位
签名	sign	string (32)	签名，详见签名生成算法
业务结果	result_code	string (16)	SUCCESS/FAIL
错误代码	err_code	string (32)	
错误代码描述	err_code_des	string (128)	错误返回的信息描述
是否重调	sub_mch_id	string (32)	子商户号

根据表 7-17 创建实体类 CloseOrderRes，如代码清单 7-43 所示。

代码清单 7-43

```
namespace WxApi.PayEntity
{
    public class CloseOrderRes:BasePayRes
    {
        public string sub_mch_id { get; set; }
    }
}
```

实体创建完毕后，在 Pay 类中添加调用接口的方法 CloseOrder，如代码清单 7-44 所示。

代码清单 7-44

```
public static CloseOrderRes CloseOrder(CloseOrder close, string key)
{
}
```



```
var url = "https://api.mch.weixin.qq.com/pay/closeorder";  
return PayRequest<CloseOrderRes>(close, key, url);  
}
```

7.11 退款 API

当交易发生之后一段时间内，由于买家或者卖家的原因需要退款时，卖家可以通过退款接口将支付款退还给买家，微信支付将在收到退款请求并且验证成功之后，按照退款规则将支付款按原路退到买家账号上。

需要注意的是，交易时间超过半年的订单无法提交退款。微信支付退款支持单笔交易分多次退款，多次退款需要提交原支付订单的商户订单号和设置不同的退款单号。一笔退款失败后重新提交，要采用原来的退款单号。总退款金额不能超过用户实际支付金额。

接口地址如下：<https://api.mch.weixin.qq.com/secapi/pay/refund>。由于调用此接口需要调用证书，因此需要在服务器端安装证书。安装证书的步骤如下。

1. 下载证书

登录微信支付商户平台 (<https://pay.weixin.qq.com>)，依次进入“账户设置”→“API安全”→“下载证书”，如图 7-14 所示。

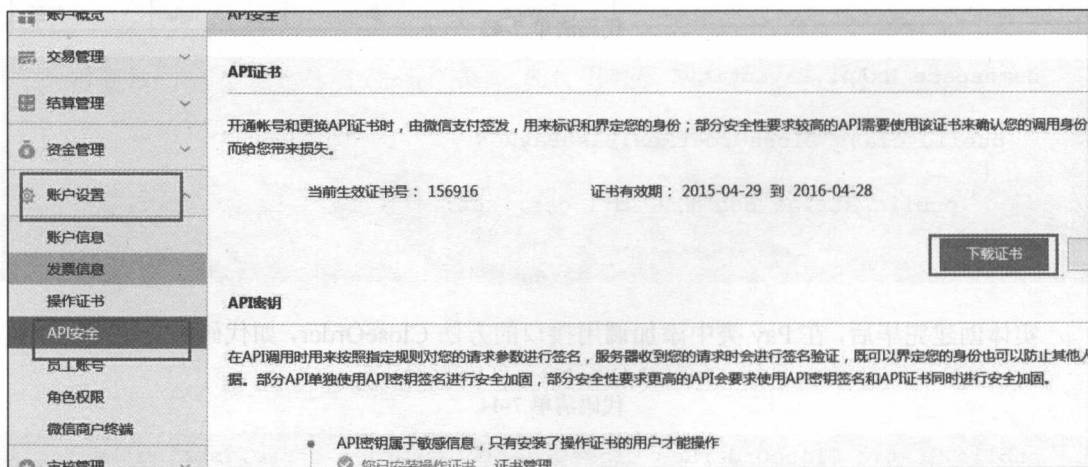


图 7-14



2. 导入证书到操作系统

证书下载完毕后进行解压。执行快捷键“win+R”，打开“运行”窗口，执行命令 mmc 打开控制台。依次单击“文件”→“添加/删除管理单元”，在打开的对话框中，在“可用的管理单元”一栏中找到“证书”后双击，打开“证书管理单元”对话框，如图 7-15 所示。

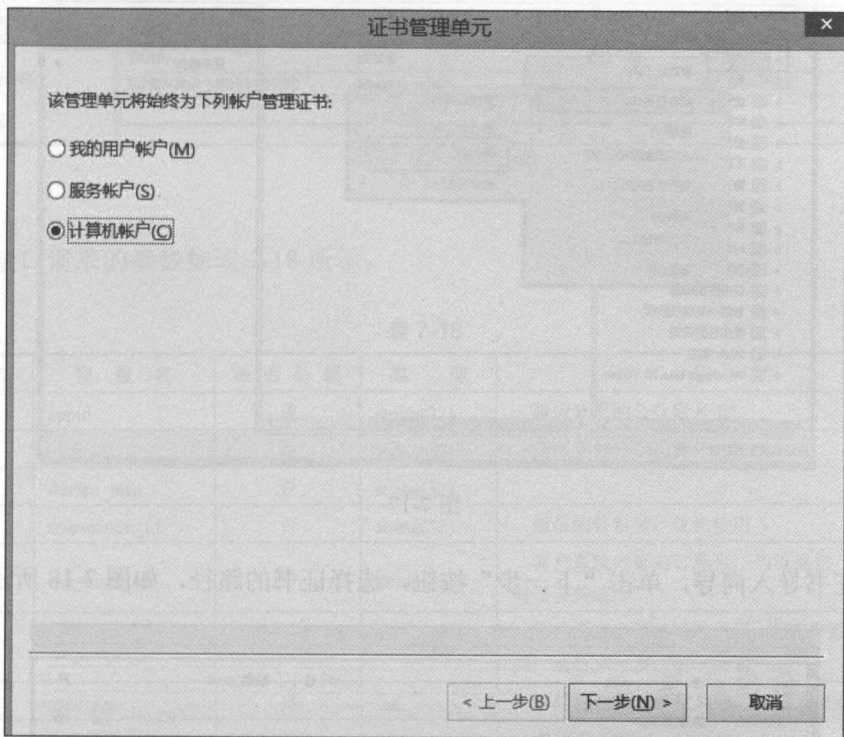


图 7-15

如图 7-15 所示，选择“计算机账户”后，依次单击“下一步”→“完成”按钮，最后单击“确定”按钮。此时，“控制台”中如图 7-16 所示。

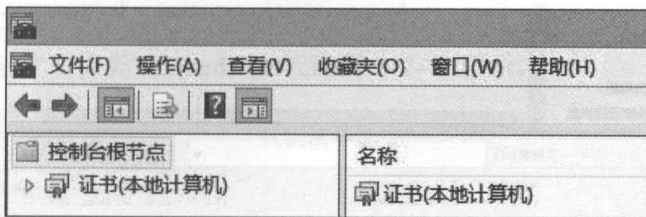


图 7-16



展开“证书”节点，右击“个人”项，在弹出的快捷菜单中选择“所有任务”→“导入”，如图 7-17 所示。

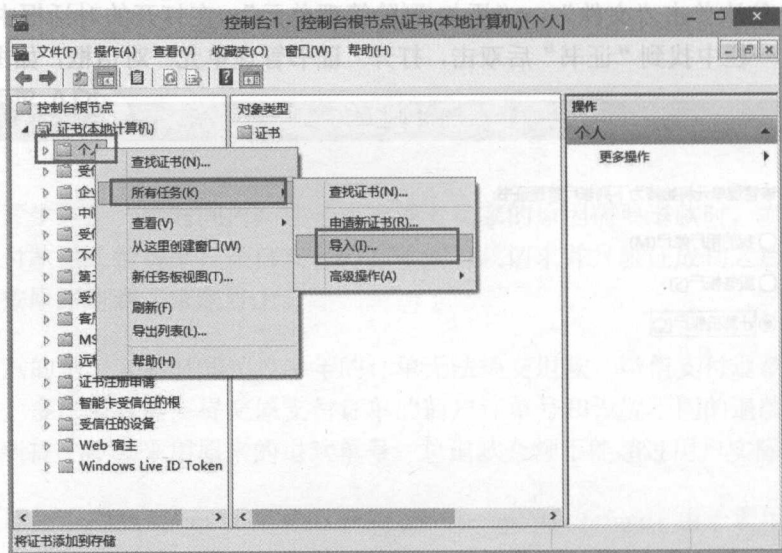


图 7-17

进入证书导入向导，单击“下一步”按钮，选择证书的路径，如图 7-18 所示。

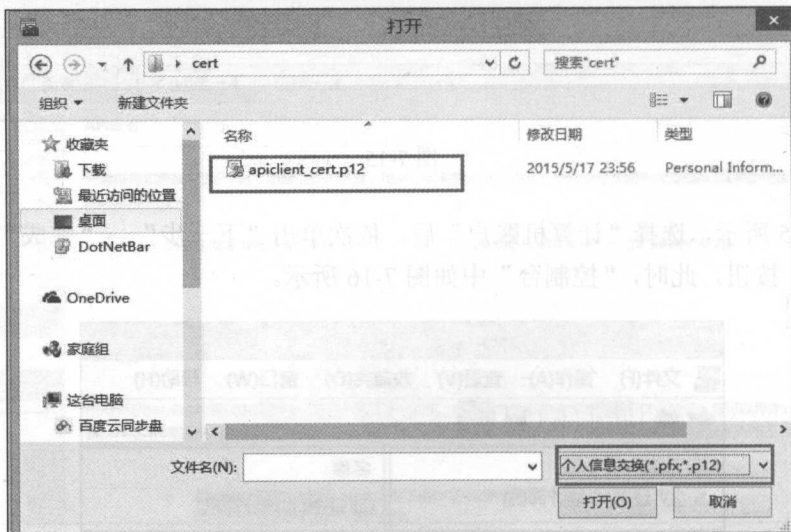


图 7-18



选择文件 `apiclient_cert.p12`，单击“下一步”按钮后输入密码（密码为商户号）。然后一路单击“下一步”按钮就 OK 了。证书导入成功后可以看到如图 7-19 所示的证书列表。至此证书安装完毕。

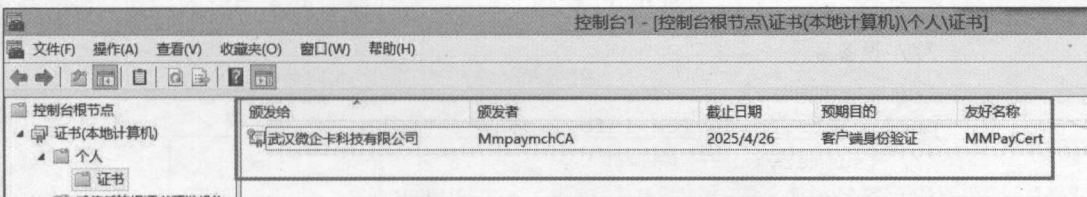


图 7-19

退款接口请求的参数如表 7-18 所示。

表 7-18

字段名	变量名	是否必填	类型	说明
公众号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
设备号	device_info	否	string(32)	
微信订单号	transaction_id	否	string(32)	微信的订单号，优先使用
商户订单号	out_trade_no	否	string(32)	商户系统内部的订单号，当没提供 transaction_id 时需要传这个
商户退款单号	out_refund_no	是	string(32)	商户系统内部的退款单号。在商户系统内部唯一。同一退款单号多次请求只退一笔
总金额	total_fee	是	int	订单总金额，单位为分，只能为整数
退款金额	refund_fee	是	int	退款总金额
货币种类	refund_fee_type	否	string(8)	
操作员	op_user_id	是	string(32)	操作员账号，默认为商户号
随机字符串	nonce_str	是	string(32)	
签名	sign	是	string(32)	

根据表 7-18 创建实体类 `ReFund`，如代码清单 7-45 所示。

代码清单 7-45

```
namespace WxApi.PayEntity
{
    /// <summary>
```



```
/// 退款请求实体
/// </summary>
public class ReFund:BasePay
{
    /// <summary>
    /// 设备号
    /// </summary>
    public string device_info { get; set; }
    /// <summary>
    /// 签名
    /// </summary>
    public string sign { get; set; }
    /// <summary>
    /// 微信订单号
    /// </summary>
    public string transaction_id { get; set; }
    /// <summary>
    /// 商户订单号
    /// </summary>
    public string out_trade_no { get; set; }
    /// <summary>
    /// 商户退款单号
    /// </summary>
    public string out_refund_no { get; set; }
    /// <summary>
    /// 总金额
    /// </summary>
    public int total_fee { get; set; }
    /// <summary>
    /// 退款金额
    /// </summary>
    public int refund_fee { get; set; }
    /// <summary>
    /// 操作员
    /// </summary>
    public string op_user_id { get; set; }
    /// <summary>
    /// 货币种类
    /// </summary>
}
```



```
public string refund_fee_type { get; set; }
```

接口调用成功后返回详细的参数列表，如表 7-19 所示。

表 7-19

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看 result_code 来判断
返回信息	return_msg	string(128)	返回信息。如非空，为错误原因
公众账号 ID	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
设备号	device_info	string(32)	终端设备号（门店号或收银设备 ID），注意：PC 网页或公众号内支付请传“WEB”
随机字符串	nonce_str	string(32)	随机字符串，不长于 32 位
签名	sign	string(32)	签名，详见签名生成算法
业务结果	result_code	string(16)	SUCCESS/FAIL
错误代码	err_code	string(32)	
错误代码描述	err_code_des	string(128)	错误返回的信息描述
商户订单号	out_trade_no	string(32)	
微信订单号	transaction_id	string(28)	
商户退款号	out_refund_no	string(32)	
微信退款号	refund_id	string(28)	
退款渠道	refund_channel	string(16)	ORIGINAL——原路退款 BALANCE——退回到余额
退款金额	refund_fee	int	
订单总金额	total_fee	int	
订单金额货币种类	fee_type	string(8)	
现金支付金额	cash_fee	int	
现金退款金额	cash_refund_fee	int	
代金券或立减优惠退款金额	coupon_refund_fee	int	代金券或立减优惠退款金额 = 订单金额 - 现金退款金额，注意：立减优惠金额不会退回
代金券或立减优惠使用数量	coupon_refund_count	int	
代金券或立减优惠 ID	coupon_refund_id	string(20)	



根据表 7-19 创建实体类 ReFundRes, 如代码清单 7-46 所示。

代码清单 7-46

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 退款请求返回的实体类
    /// </summary>
    public class ReFundRes:BasePayRes
    {
        /// <summary>
        /// 设备号
        /// </summary>
        public string device_info { get; set; }
        /// <summary>
        /// 微信订单号
        /// </summary>
        public string transaction_id { get; set; }
        /// <summary>
        /// 商户订单号
        /// </summary>
        public string out_trade_no { get; set; }
        /// <summary>
        /// 商户退款单号
        /// </summary>
        public string out_refund_no { get; set; }
        /// <summary>
        /// 微信退款单号
        /// </summary>
        public string refund_id { get; set; }
        /// <summary>
        /// 退款渠道 ORIGINAL——原路退款, BALANCE——退回到余额
        /// </summary>
        public string refund_channel { get; set; }
        /// <summary>
        /// 退款金额
        /// </summary>
        public int refund_fee { get; set; }
```




```
/// <summary>
/// 订单总金额
/// </summary>
public int total_fee { get; set; }
/// <summary>
/// 订单金额货币种类
/// </summary>
public string fee_type{ get; set; }
/// <summary>
/// 现金支付金额
/// </summary>
public int cash_fee { get; set; }
/// <summary>
/// 现金退款金额
/// </summary>
public int cash_refund_fee { get; set; }
/// <summary>
/// 代金券或立减优惠退款金额
/// </summary>
public int coupon_refund_fee { get; set; }
/// <summary>
/// 代金券或立减优惠使用数量
/// </summary>
public int coupon_refund_count{ get; set; }
/// <summary>
/// 代金券或立减优惠 ID
/// </summary>
public string coupon_refund_id { get; set; }
}
}
```

实体创建完毕后，在 Pay 类中添加实现此接口的方法 ReFund，如代码清单 7-47 所示。

代码清单 7-47

```
public static ReFundRes ReFund(ReFund reFund, string key, string certpath,
string certpwd)
{
    var url = "https://api.mch.weixin.qq.com/secapi/pay/refund";
    return PayRequest<ReFundRes>(reFund, key,url,certpath,certpwd);
}
```




由于退款有一定延时,用零钱支付的退款 20 分钟内到账,银行卡支付的退款 3 个工作日内到账;因此提交退款申请后,需要通过“查询退款接口”查询退款状态。

查询退款的接口地址是 <https://api.mch.weixin.qq.com/pay/refundquery>。

请求参数如表 7-20 所示。

表 7-20

字段名	变量名	是否必填	类型	说明
公众号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
设备号	device_info	否	string(32)	
微信订单号	transaction_id	否	string(28)	微信的订单号, 优先使用
商户订单号	out_trade_no	否	string(32)	商户系统内部的订单号, 当没提供 transaction_id 时需要传这个
商户退款单号	out_refund_no	否	string(32)	
微信退款单号	refund_id	否	string(28)	refund_id、out_refund_no、out_trade_no、transaction_id 四个参数必填一个。如果同时存在, 则优先级为: refund_id>out_refund_no>transaction_id>out_trade_no
随机字符串	nonce_str	是	string(32)	
签名	sign	是	string(32)	

根据表 7-20 创建实体类 QueryRefund, 如代码清单 7-48 所示。

代码清单 7-48

```
namespace WxApi.PayEntity
{
    public class QueryRefund:BasePay
    {
        public string transaction_id { get; set; }
        public string out_trade_no { get; set; }
        public string device_info { get; set; }
        public string out_refund_no { get; set; }
        public string refund_id { get; set; }
        public string sign { get; set; }
    }
}
```



接口请求成功后，返回的参数列表如表 7-21 所示。

表 7-21

字段名	变量名	类型	说明
返回状态码	return_code	string(16)	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看 trade_state 来判断
返回信息	return_msg	string(128)	返回信息。如非空，为错误原因
以下字段在 return_code 为 SUCCESS 的时候有返回			
公众账号 ID	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
随机字符串	nonce_str	string(32)	
签名	sign	string(32)	
业务结果	result_code	string(16)	SUCCESS/FAIL
错误代码	err_code	string(32)	
错误代码描述	err_code_des	string(128)	
以下字段在 return_code 和 result_code 都为 SUCCESS 的时候有返回			
设备号	device_info	string(32)	微信支付分配的终端设备号
总金额	total_fee	int	订单总金额，单位为分
货币种类	fee_type	string(8)	默认人民币：CNY
现金支付金额	cash_fee	int	
现金支付货币类型	cash_fee_type	string(16)	
退款金额	refund_fee	int	
代金券或立减优惠退款金额	coupon_refund_fee	int	“代金券或立减优惠”金额 ≤ 订单总金额，订单总金额 - “代金券或立减优惠”金额 = 现金支付金额
退款笔数	refund_count	int	
商户退款单号	out_refund_no_\$n	string(32)	
微信退款单号	refund_id_\$n	string(28)	
退款渠道	refund_channel_\$n	string(16)	
退款金额	refund_fee_\$n	int	
货币种类	fee_type_\$n	string(8)	
代金券或立减优惠退款金额	coupon_refund_fee_\$n	int	
代金券或立减优惠使用数量	coupon_refund_count_\$n	int	
退款状态	refund_status_\$n	string(16)	
微信支付订单号	transaction_id	string(32)	
商户订单号	out_trade_no	string(32)	商户系统内部的订单号，当没提供 transaction_id 时需要传这个



根据表 7-21 创建实体类 QueryRefundRes，如代码清单 7-49 所示。

代码清单 7-49

```
using System.Collections.Generic;

namespace WxApi.PayEntity
{
    public class QueryRefundRes:BasePayRes
    {
        public string transaction_id { get; set; }
        public string out_trade_no { get; set; }
        public string total_fee { get; set; }
        public string fee_type { get; set; }
        public string cash_fee { get; set; }
        public string cash_fee_type { get; set; }
        public string refund_fee { get; set; }
        public string coupon_refund_fee { get; set; }
        public string refund_count { get; set; }
        public List<RefundInfo> _Infos { get; set; }
    }
    public class RefundInfo
    {
        public string out_refund_no { get; set; }
        public string refund_id { get; set; }
        public string refund_channel { get; set; }
        public string refund_fee { get; set; }
        public string fee_type { get; set; }
        public string coupon_refund_fee { get; set; }
        public string coupon_refund_count { get; set; }
        public string refund_status { get; set; }
    }
}
```

实体创建完毕后，在 Pay 类中添加实现此接口的方法 QueryReFund，如代码清单 7-50 所示。

代码清单 7-50

```
public static QueryRefundRes QueryReFund(QueryRefund refund, string key)
{
```



```
var url = "https://api.mch.weixin.qq.com/pay/refundquery";  
return PayRequest<QueryRefundRes>(refund, key, url);  
}
```

请注意：假如退款的订单使用了代金券或满减优惠，则在退款时会出现两个下标的字段，在序列化、反序列化的时候处理异常会很麻烦，且官方文档关于两个下标字段的说明不是很明白，笔者也没看出实际的意义，所以在反序列化的时候直接移除了两个下标的字段。这样处理会有一个问题，那就是在验证签名的时候会和返回的不一致，不过一般情况下是不影响业务逻辑的。期待腾讯官方能完善一下开发文档。（笔者表示无力“吐槽”：明明可以使用列表，却偏偏使用下标……）

7.12 商户营销与支付工具

7.12.1 代金券或立减优惠

微信支付代金券业务是基于微信支付，为了协助商户方便地实现营销优惠措施的一种工具。针对部分有开发能力的商户，微信支付提供通过 API 接口实现运营代金券的功能。

现阶段，微信仅提供了发放代金券和查询代金券信息接口，未提供创建代金券或立减优惠的接口（截至 2015 年 5 月 19 日），所以在调用发放接口之前，需要商户在商户后台进行创建后并激活。创建好的代金券如图 7-20 所示。

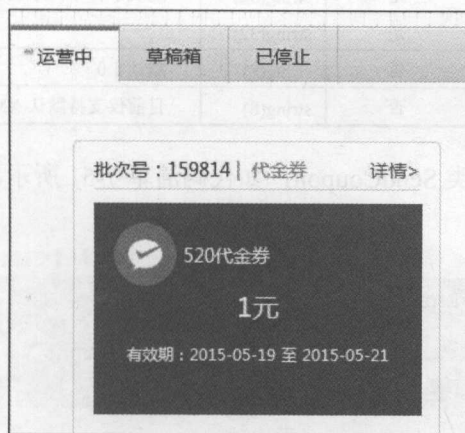


图 7-20



关于代金券和立减优惠的介绍请参考本书下载资源中的《微信支付代金券及立减优惠系统操作手册》和《微信支付营销工具介绍》。

发放代金券的接口地址如下：

https://api.mch.weixin.qq.com/mmpaymkttransfers/send_coupon

HTTP 请求方式：POST，需验证签名。

请求参数如表 7-22 所示。

表 7-22

字段名	变量名	是否必填	类型	说明
代金券批次 id	coupon_stock_id	是	string	代金券批次 id
openid 记录数	openid_count	是	int	openid 记录数（目前支持 num=1）
商户单据号	partner_trade_no	是	string	商户此次发放凭据号（格式：商户 id+日期+流水号），商户侧需保持唯一性
用户 openid	openid	是	string	openid 信息
公众账号 ID	appid	是	string(32)	微信分配的公众账号 ID
商户号	mch_id	是	string(32)	微信支付分配的商户号
子商户号	sub_mch_id	否	string(32)	微信支付分配的子商户号，受理模式下必填
操作员	op_user_id	是	string(32)	操作员账号，默认为商户号。可在商户平台配置操作员对应的 API 权限
设备号	device_info	否	string(32)	微信支付分配的终端设备号
随机字符串	nonce_str	是	string(32)	随机字符串，不长于 32 位
签名	sign	是	string(32)	
协议版本	version	否	string(8)	默认 1.0
协议类型	type	否	string(8)	目前仅支持默认 XML

根据表 7-22 创建实体类 SendCoupon，如代码清单 7-51 所示。

代码清单 7-51

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 发放代金券实体
    /// </summary>
    public class SendCoupon : BasePay
```




```
{

    /// <summary>
    /// 代金券批次 ID
    /// </summary>
    public string coupon_stock_id { get; set; }
    /// <summary>
    /// openid 记录数 (目前支持 num=1)
    /// </summary>
    public int openid_count { get; set; }
    /// <summary>
    /// 商户此次发放凭据号 (格式: 商户 id+日期+流水号), 商户侧需保持唯一性
    /// </summary>
    public string partner_trade_no { get; set; }
    public string openid { get; set; }
    /// <summary>
    /// 子商户号
    /// </summary>
    public string sub_mch_id { get; set; }
    /// <summary>
    /// 操作员
    /// </summary>
    public string op_user_id { get; set; }
    /// <summary>
    /// 设备号
    /// </summary>
    public string device_info { get; set; }

    public string sign { get; set; }
    /// <summary>
    /// 协议版本 默认 1.0
    /// </summary>
    public string version { get; set; }
    /// <summary>
    /// 协议类型 XML 【目前仅支持默认 XML】
    /// </summary>
    public string type { get; set; }
}
```

接口请求成功后, 返回的参数列表如表 7-23 所示。



表 7-23

字段名	变量名	类型	说明
基础参数列表，同其他接口返回信息，如统一下单接口			
商户单据号	partner_trade_no	string	商户此次发放凭据号（格式：商户 id+日期+流水号），商户侧需保持唯一性
公众账号 ID	appid	string(32)	微信分配的公众账号 ID
商户号	mch_id	string(32)	微信支付分配的商户号
子商户号	sub_mch_id	string(32)	微信支付分配的子商户号，受理模式下必填
设备号	device_info	string(32)	微信支付分配的终端设备号
用户 openid	openid	string	
随机字符串	nonce_str	string(32)	随机字符串，不长于 32 位
签名	sign	string(32)	
代金券批次 id	coupon_stock_id	string	
返回记录数	resp_count	int	
成功记录数	success_count	int	1 或者 0
失败记录数	failed_count	int	1 或者 0
用户标识	openid	string	
返回码	ret_code	string	
代金券 id	coupon_id	string	对一个用户成功发放代金券，则返回代金券 id（即 ret_code 为 SUCCESS 的时候）；如果 ret_code 为 FAILED，则为空串""
返回信息	ret_msg	string	
协议版本	version	string(8)	默认 1.0
协议类型	type	string(8)	目前仅支持默认 XML

根据表 7-23 创建实体类 SendCouponRes，如代码清单 7-52 所示。

代码清单 7-52

```
namespace WxApi.PayEntity
{
    public class SendCouponRes : BasePayRes
    {
        public string appid { get; set; }
        public string mch_id { get; set; }
        public string sub_mch_id { get; set; }
        public string device_info { get; set; }
        public string nonce_str { get; set; }
        public string coupon_stock_id { get; set; }
        public int resp_count { get; set; }
    }
}
```



```

public int success_count { get; set; }
public int failed_count { get; set; }
public string openid { get; set; }
public string ret_code { get; set; }
public string coupon_id { get; set; }
public string ret_msg { get; set; }
}
}

```

实体创建完毕后，在 Pay 类中添加接口调用的方法 SendCoupon，如代码清单 7-53 所示。

代码清单 7-53

```

public static SendCouponRes SendCoupon(SendCoupon coupon, string key, string
certpath, string certpwd)
{
    var url = "https://api.mch.weixin.qq.com/mmpaymkttransfers/send_coupon";
    return PayRequest<SendCouponRes>(coupon, key, url, certpath, certpwd);
}

```

代金券成功发送后，将通过微信支付公众号发送给用户，如图 7-21 所示。用户领取后，此代金券将添加到微信卡包中，当用户在满足此代金券使用条件的情况下，在付款的时候自动使用。

7.12.2 现金红包

春节期间，微信红包以其独特的魅力、优秀的用户体验和安全的支付环境，一经推出即受到了广大用户的热烈欢迎。现在微信支付现金红包向微信支付商户开放，具体能力如下：

(1) 商户调用接口时，通过指定发送对象以及发送金额的方式发放红包。这样的方式，允许商户灵活地应用于各种各样的活动场景。

(2) 领取到红包后，用户的资金直接进入微信零钱，避

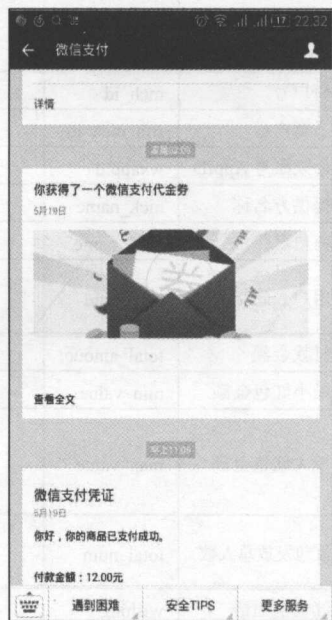


图 7-21



免繁复的领奖流程，可带给用户微信支付原生的流畅体验。

注意：微信红包每分钟发送红包的数量不得超过 1800 个；北京时间 0:00-8:00 不触发红包接口。每个红包金额介于[1.00 元, 200.00 元]之间；同一个红包只能发送给一个用户（如果以上规则不满足你的需求，则请发邮件至 wxhongbao@tencent.com 获取升级指引）。

接口地址如下：

<https://api.mch.weixin.qq.com/mmpaymkttransfers/sendredpack>

HTTP 请求方式：POST，需要证书。

请求参数如表 7-24 所示。

表 7-24

字段名	变量名	是否必填	类型	说明
随机字符串	nonce_str	是	string(32)	随机字符串，不长于 32 位
签名	sign	是	string(32)	详见签名生成算法
商户订单号	mch_billno	是	string(28)	商户订单号（每个订单号必须唯一） 组成：mch_id+yyyymmdd+10 位一天内不能重复的数字。 接口根据商户订单号支持重入，如出现超时可再调用
商户号	mch_id	是	string(32)	微信支付分配的商户号
子商户号	sub_mch_id	否	string(32)	微信支付分配的子商户号，受理模式下必填
公众账号 AppID	wxappid	是	string(32)	商户 AppID
提供方名称	nick_name	是	string(32)	提供方名称
商户名称	send_name	是	string(32)	红包发送者名称
用户 openid	re_openid	是	string(32)	接收红包的用户 用户在 wxappid 下的 openid
付款金额	total_amount	是	int	付款金额，单位为分
最小红包金额	min_value	是	int	最小红包金额，单位为分
最大红包金额	max_value	是	int	最大红包金额，单位为分 (最小金额等于最大金额：min_value=max_value = total_amount)
红包发放总人数	total_num	是	int	红包发放总人数 total_num=1
红包祝福语	wishing	是	string(128)	红包祝福语
Ip 地址	client_ip	是	string(15)	调用接口的机器 Ip 地址



续表

字段名	变量名	是否必填	类型	说明
活动名称	act_name	是	string(32)	活动名称
备注	remark	是	string(256)	备注信息
商户 logo 的 URL	logo_imgurl	否	string(128)	商户 logo 的 URL
分享文案	share_content	否	string(256)	分享文案
分享链接	share_url	否	string(128)	分享链接
分享的图片	share_imgurl	否	string(128)	分享的图片 URL

根据表 7-24 创建实体类 RedPack，如代码清单 7-54 所示。

代码清单 7-54

```
namespace WxApi.PayEntity
{
    public class RedPack
    {
        /// <summary>
        /// 随机字符串
        /// </summary>
        public string nonce_str { get; set; }
        /// <summary>
        /// 签名
        /// </summary>
        public string sign { get; set; }
        /// <summary>
        /// 商户订单号
        /// </summary>
        public string mch_billno { get; set; }
        /// <summary>
        /// 商户号
        /// </summary>
        public string mch_id { get; set; }
        /// <summary>
        /// 子商户号
        /// </summary>
        public string sub_mch_id { get; set; }
        /// <summary>
        /// 公众账号 AppID
    }
}
```




```
/// </summary>
public string wxappid { get; set; }
/// <summary>
/// 提供方名称
/// </summary>
public string nick_name { get; set; }
/// <summary>
/// 商户名称
/// </summary>
public string send_name { get; set; }
/// <summary>
/// 用户 openid
/// </summary>
public string re_openid { get; set; }
/// <summary>
/// 付款金额
/// </summary>
public int total_amount { get; set; }
/// <summary>
/// 最小红包金额
/// </summary>
public int min_value { get; set; }
/// <summary>
/// 最大红包金额,最小金额等于最大金额: min_value=max_value =total_amount
/// </summary>
public int max_value { get; set; }
/// <summary>
/// 红包发放总人数
/// </summary>
public int total_num { get; set; }
/// <summary>
/// 红包祝福语
/// </summary>
public string wishing{ get; set; }
/// <summary>
/// IP 地址
/// </summary>
public string client_ip{ get; set; }
/// <summary>
/// 活动名称
```



```

    /// </summary>
    public string act_name { get; set; }
    /// <summary>
    /// 备注
    /// </summary>
    public string remark { get; set; }
    /// <summary>
    /// 商户 logo 的 URL
    /// </summary>
    public string logo_imgurl { get; set; }
    /// <summary>
    /// 分享文案
    /// </summary>
    public string share_content{ get; set; }
    /// <summary>
    /// 分享链接
    /// </summary>
    public string share_url { get; set; }
    /// <summary>
    /// 分享的图片
    /// </summary>
    public string share_imgurl{ get; set; }
}

```

请求成功后返回的参数列表如表 7-25 所示。

表 7-25

字段名	变量名	类型	说明
基础参数列表，同其他接口返回信息，如统一下单接口			
签名	sign	string(32)	
商户订单号	mch_billno	string(28)	商户订单号
商户号	mch_id	string(32)	微信支付分配的商户号
公众账号 appid	wxappid	string(32)	
用户 openid	re_openid	string(32)	
付款金额	total_amount	int	付款金额，单位为分



根据表 7-25 创建实体类 RedPackRes，如代码清单 7-55 所示。

代码清单 7-55

```
namespace WxApi.PayEntity
{
    public class RedPackRes:BasePayRes
    {
        public string mch_billno{ get; set; }
        public string mch_id{ get; set; }
        public string wxappid{ get; set; }
        public string re_openid{ get; set; }
        public int total_amount{ get; set; }
        public string send_listid { get; set; }
        public string send_time { get; set; }
    }
}
```

实体创建完毕后，在 Pay 类中添加接口调用的方法 SendRePack，如代码清单 7-56 所示。

代码清单 7-56

```
public static RedPackRes SendRePack(RedPack redPack, string key, string
certpath, string certpwd)
{
    var url = "https://api.mch.weixin.qq.com/mmpaymkttransfers/
sendredpack";
    return PayRequest<RedPackRes>(redPack, key, url, certpath, certpwd);
}
```

7.12.3 企业付款

企业付款业务是基于微信支付商户平台的资金管理能力，为了协助商户方便地实现企业向个人付款，针对部分有开发能力的商户，提供通过 API 完成企业付款的功能。比如目前的保险行业向客户退保、给付、理赔。

企业付款将使用商户的可用余额，需确保可用余额充足。注意：与商户微信支付收款资金并非同一账户，需要单独充值。



接口地址如下:

<https://api.mch.weixin.qq.com/mmpaymkttransfers/promotion/transfers>

HTTP 请求方式: POST, 需要证书。

请求参数如表 7-26 所示。

表 7-26

字段名	变量名	是否必填	类型	说明
随机字符串	nonce_str	是	string(32)	随机字符串, 不长于 32 位
公众账号 AppID	mch_appid	是	string	商户 AppID
商户号	mchid	是	string(32)	微信支付分配的商户号
子商户号	sub_mch_id	否	string(32)	微信支付分配的子商户号, 受理模式下必填
设备号	device_info	否	string(32)	微信支付分配的终端设备号
随机字符串	nonce_str	是	string(32)	随机字符串, 不长于 32 位
签名	sign	是	string(32)	生成签名方式可查看 7.2.3 节
商户订单号	partner_trade_no	是	string	商户订单号, 需保持唯一性
用户 openid	openid	是	string	商户 AppID 下, 某用户的 openid
校验用户姓名选项	check_name	是	string	NO_CHECK: 不校验真实姓名 FORCE_CHECK: 强校验真实姓名 (未实名认证的用户会校验失败, 无法转账) OPTION_CHECK: 针对已实名认证的用户才校验真实姓名 (未实名认证的用户不校验, 可以转账成功)
收款用户姓名	re_user_name	可选	string	收款用户真实姓名。如果 check_name 设置为 FORCE_CHECK 或 OPTION_CHECK, 则必填用户真实姓名
金额	amount	是	UInt64_t	企业付款金额, 单位为分
企业付款描述信息	desc	是	string	企业付款操作说明信息。必填
Ip 地址	spbill_create_ip	是	string(32)	调用接口的机器 Ip 地址

根据表 7-26 创建实体类 EPayment, 如代码清单 7-57 所示。

代码清单 7-57

```
namespace WxApi.PayEntity
{
    public class EPayment
```




```
{  
    /// <summary>  
    /// 公众账号 appid  
    /// </summary>  
    public string mch_appid { get; set; }  
    /// <summary>  
    /// 商户号  
    /// </summary>  
    public string mchid { get; set; }  
    /// <summary>  
    /// 子商户号  
    /// </summary>  
    public string sub_mch_id { get; set; }  
    /// <summary>  
    /// 设备号  
    /// </summary>  
    public string device_info { get; set; }  
    /// <summary>  
    /// 随机字符串  
    /// </summary>  
    public string nonce_str { get; set; }  
    public string sign { get; set; }  
    /// <summary>  
    /// 商户订单号  
    /// </summary>  
    public string partner_trade_no { get; set; }  
    public string openid { get; set; }  
    /// <summary>  
    /// NO_CHECK: 不校验真实姓名 FORCE_CHECK: 强校验真实姓名 (未实名认证的用户  
    /// 会校验失败, 无法转账) OPTION_CHECK: 针对已实名认证的用户才校验真实姓名 (未  
    /// 实名认证的用户不校验, 可以转账成功)  
    /// </summary>  
    public CheckNameOption check_name { get; set; }  
    /// <summary>  
    /// 收款用户真实姓名。如果 check_name 设置为 FORCE_CHECK 或 OPTION_CHECK,  
    /// 则必填用户真实姓名  
    /// </summary>  
    public string re_user_name { get; set; }  
    /// <summary>  
    /// 企业付款金额, 单位为分
```




```

    /// </summary>
    public int amount { get; set; }
    /// <summary>
    /// 企业付款描述信息
    /// </summary>
    public string desc { get; set; }
    /// <summary>
    /// 调用接口的机器 IP 地址
    /// </summary>
    public string spbill_create_ip { get; set; }
}
public enum CheckNameOption
{
    /// <summary>
    /// 不校验真实姓名
    /// </summary>
    NO_CHECK,
    /// <summary>
    /// 强校验真实姓名（未实名认证的用户会校验失败，无法转账）
    /// </summary>
    FORCE_CHECK,
    /// <summary>
    /// 针对已实名认证的用户才校验真实姓名（未实名认证的用户不校验，可以转账成功）
    /// </summary>
    OPTION_CHECK
}
}

```

接口请求成功后，返回的参数列表如表 7-27 所示。

表 7-27

字段名	变量名	类型	说明
基础参数列表，同其他接口返回信息，如统一下单接口			
商户 appid	mch_appid	string	商户 appid
商户号	mchid	string(32)	微信支付分配的商户号
设备号	device_info	string(32)	微信支付分配的终端设备号
随机字符串	nonce_str	string(32)	随机字符串，不长于 32 位
签名	sign	string(32)	
商户订单号	partner_trade_no	string(32)	商户订单号，需保持唯一性
微信订单号	payment_no	string	企业付款成功，返回的微信订单号
微信成功时间	payment_time	String	企业付款成功时间



根据表 7-27 创建实体类 EPaymentRes，如代码清单 7-58 所示。

代码清单 7-58

```
namespace WxApi.PayEntity
{
    /// <summary>
    /// 企业付款实体
    /// </summary>
    public class EPaymentRes : BasePayRes
    {
        public string mch_appid { get; set; }
        public string mchid { get; set; }
        public string device_info { get; set; }
        public string nonce_str { get; set; }
        public string partner_trade_no { get; set; }
        public string payment_no { get; set; }
        public string payment_time { get; set; }
    }
}
```

实体创建完毕后，在 Pay 类中添加接口调用的方法 EPayment，如代码清单 7-59 所示。

代码清单 7-59

```
public static EPaymentRes EPayment(EPayment payment, string key, string
certpath, string certpwd)
{
    var url = "https://api.mch.weixin.qq.com/mmpaymkttransfers/promotion/
transfers";
    return PayRequest<EPaymentRes>(payment, key, url, certpath, certpwd);
}
```

第 8 章 微信小店开发

8.1 微信小店的开通与搭建

微信小店是基于微信公众平台打造的原生电商模式，包括商品管理、订单管理、货架管理、维权管理等功能，开发者可使用接口批量添加商品，快速开店。微信小店的开通方式简单，只要已经是获得了微信认证且开通微信支付的服务号即可开通。

开通小店后，首先是添加商品，如图 8-1 所示。

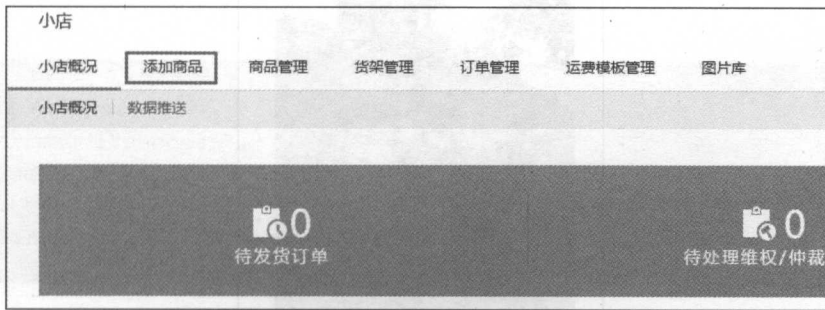


图 8-1

单击“添加商品”按钮后，进入添加商品流程。首先选择商品所属的分类。选择分类后，进入编辑产品详细信息页面。选择商品的属性、输入库存信息、物流、售后等信息后，单击“保存”按钮即可添加成功，如图 8-2 所示。

添加商品后，需要提供一种方便展示的方式，让用户方便快捷地找到自己所需的商品。



为此，微信引入了货架的概念。通俗地讲，货架就是展示商品的展台。商家可以添加多个货架，并可组合多种展示方式。货架添加成功后，可下载货架的二维码或获取货架的 URL，引导用户扫描二维码或打开 URL，如图 8-3 所示。

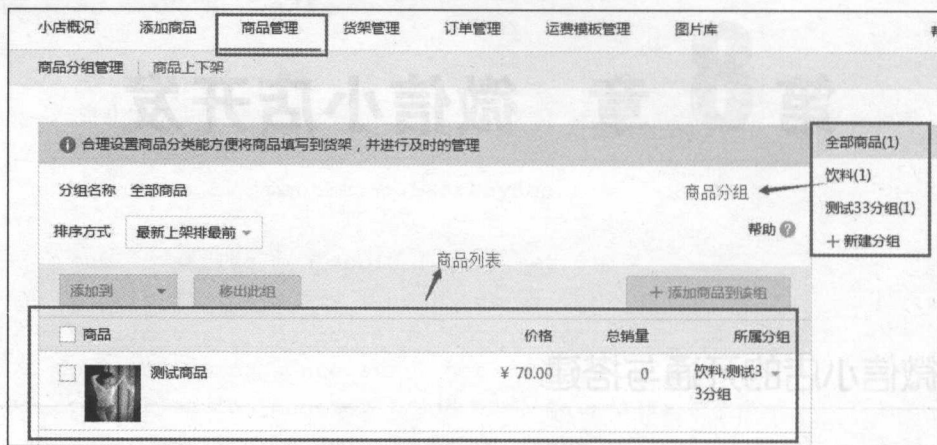


图 8-2

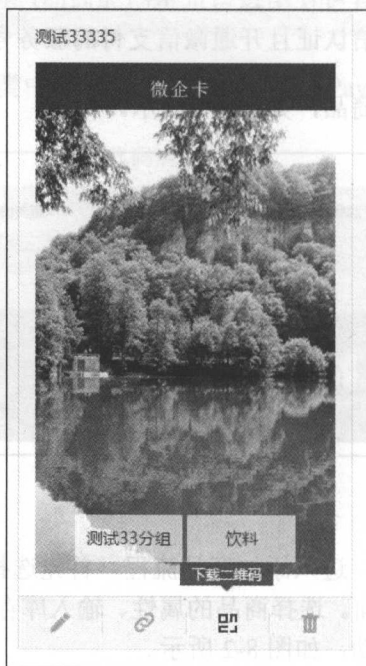


图 8-3



普通用户可直接通过微信公众平台官方后台管理小店，开发者则可以通过开发接口来实现更灵活的小店运营。

8.2 商品管理

商品管理主要包括商品的增删查改和上下架操作。在添加商品的过程中，需要先选择商品所属的分类，而不同分类下的商品属性和 SKU 可能也是不一样的，所以在此过程中需要先获取分类的列表，然后根据所选的分类去获取此分类的所有属性和 SKU 信息。另外，商品的图片和微信小店其他接口所需要的图片均需要调用指定的接口来获取图片的 URL。上传图片的接口地址如下：

```
https://api.weixin.qq.com/merchant/common/upload_img?access_token=ACCESS_TOKEN
& filename = test.png
```

在上述接口地址中，filename 表示的是图片的文件名。HTTP 请求方式是：POST。在调用的过程中，将文件转换成流（Stream），然后写入请求流（注意：此接口的调用方式是 POST，和 3.2 节中素材管理接口的 POST/FORM 方式是不一样的）。调用成功后，返回的数据格式如代码清单 8-1 所示。

代码清单 8-1

```
{
    "errcode":0,
    "errmsg":"success",
    "image_url":
    "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl2ib14JWwnW3icSJGqecVtRiaPxwWE
    Ir99eYYL6AAAp1YBo12CpQTXFH6InyQWXITLvU4CU7kic4PcoXA/0"
}
```

具体实现步骤如下。

第一步，在 WxApi 项目的 Utils 类中添加 HttpPost 方法的重载。如代码清单 8-2 所示，此方法有两个参数，url 表示接口的地址，stream 表示要上传的数据流。

代码清单 8-2

```
public static string HttpPost(string url, Stream stream)
```




```
{  
    //当请求为 https 时, 验证服务器证书  
    ServicePointManager.ServerCertificateValidationCallback = new  
RemoteCertificateValidationCallback((a, b, c, d) => { return true; });  
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);  
    request.Method = "POST";  
    request.ContentType = "application/x-www-form-urlencoded";  
    request.Accept = "*/*";  
    request.Timeout = 15000;  
    request.AllowAutoRedirect = false;  
    string responseStr = "";  
    using (var reqstream = request.GetRequestStream())  
    {  
        stream.Position = 0L;  
        stream.CopyTo(reqstream);  
    }  
    using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())  
    {  
        using (StreamReader reader = new  
StreamReader(response.GetResponseStream(), Encoding.UTF8))  
        {  
            responseStr = reader.ReadToEnd(); //获取响应  
        }  
    }  
    return responseStr;  
}
```

第二步, 和第一步一样, 添加 `PostResult` 方法的重载, 如代码清单 8-3 所示。

代码清单 8-3

```
public static T PostResult<T>(Stream stream, string url)  
{  
    var retdata = HttpPost(url, stream);  
    return JsonConvert.DeserializeObject<T>(retdata);  
}
```

第三步, 添加方法 `GetFileStreamInfo`, 其功能是根据文件物理路径或网络路径获取文件流和文件名, 如代码清单 8-4 所示。



代码清单 8-4

```
public static FileStreamInfo GetFileStreamInfo(string filepath)
{
    var fsi = new FileStreamInfo();
    if (!filepath.Contains("http")) //判断是否是网络路径
    {
        //获取文件流
        using (var fs = new FileStream(filepath, FileMode.Open, FileAccess.
Read))
        {
            fs.Position = 0L;
            fs.CopyTo(fsi);
        }
    }
    else
    {
        //获取文件流
        using (var client = new WebClient())
        {
            var bytes = client.DownloadData(filepath);
            fsi.Write(bytes, 0, bytes.Length);
        }
    }
    fsi.FileName = Path.GetFileName(filepath); //获取文件名
    return fsi;
}
```

第四步，封装接口返回信息实体，如代码清单 8-5 所示。

代码清单 8-5

```
namespace WxApi.ReceiveEntity.Shop
{
    public class PicInfo:ErrorEntity
    {
        public string image_url { get; set; }
    }
}
```



最后，在 WxApi 项目中添加文件夹 Shop，和微信小店相关的方法都封装在此文件夹中。在文件夹中添加类 Common，将调用上传图片的方法添加到此类中，如代码清单 8-6 所示。

代码清单 8-6

```
public static PicInfo UploadImg(string filepath, string accessToken)
{
    var fsi = Utils.GetFileStreamInfo(filepath);
    var url =
        string.Format("https://api.weixin.qq.com/merchant/common/upload_img?
        access_token={0}&filename={1}", accessToken, fsi.FileName);
    return Utils.PostResult<PicInfo>(fsi, url);
}
```

8.2.1 获取指定分类的所有子分类

在添加商品时，需要选择商品所属的分类，如图 8-4 所示。

你当前选择的是：食品/茶叶/特产/滋补品 > 果汁/饮料/牛奶制品 > 茶饮料（商品上架后不可修改，请谨慎选择）

一级分类	二级分类	三级分类
请输入类目名称	请输入类目名称	请输入类目名称
IP卡/网络电话/手机号码	饼干/糕点/膨化食品	茶饮料
MP3/MP4/录音笔/电子书	茶叶/普洱/花草茶	功能饮料
VB2C	调味品/烘焙原料	谷类坚果饮料
ZIPPO/瑞士军刀/眼镜/礼品	方便速食/罐头	果蔬汁
办公/电子辞典/文具	果汁/饮料/牛奶制品	果味饮料
保健食品	海味零食/鱿鱼丝	含乳饮料
保险	海鲜/水产品/制品	凉茶
笔记本/台式/一体/平板电脑	酒类制品	奶酪/乳制品
彩妆/香水/美妆工具	蜜饯/果脯/坚果/炒货	酸梅汁
成人用品/避孕/计生用品	南北干货/肉类干货	碳酸饮料
宠物/宠物食品及用品	藕粉/麦片/天然粉/冲饮品	饮用水
电脑硬件/周边/网络设备	肉类零食/牛肉干/豆腐干	
电玩/游戏机/配件/游戏软件	生肉/肉制品	
电影/电视/音乐/曲艺	食品提货券	
国货精品数码	熟食/凉菜/私房菜	

图 8-4



接口地址如下:

`https://api.weixin.qq.com/merchant/category/getsub?access_token=ACCESS_TOKEN`

HTTP 请求方式: POST。

请求示例: `{"cate_id": 537874913}`。

`cate_id` 表示的是父分类 ID (根节点分类 ID 为 1)。

请求成功后, 返回的数据包如代码清单 8-7 所示。

代码清单 8-7

```
{
  "errcode": 0,
  "errmsg": "success",
  "cate_list": [
    {
      "id": "537074292",
      "name": "数码相机"
    },
    {
      "id": "537074293",
      "name": "家用摄像机"
    },
  ]
}
```

在代码实现的过程中, 首先根据上述代码创建实体类。代码清单 8-8 所示的是基础实体, 映射只包含 ID 和 Name 两个属性的对象。

代码清单 8-8

```
namespace WxApi.ReceiveEntity.Shop
{
    public class BaseEntity
    {
        public string id { get; set; }
        public string name { get; set; }
    }
}
```




然后创建类 `GoodsCategoryList`，此类就是调用获取子分类接口的返回实体，如代码清单 8-9 所示。

代码清单 8-9

```
public class GoodsCategoryList : ErrorEntity
{
    public List<BaseEntity> cate_list { get; set; }
}
```

实体创建完毕后，在 `Shop` 文件夹中添加类 `GoodsManage`，用于封装商品管理相关的方法。在此类中添加方法 `GetChildCategory`，如代码清单 8-10 所示。

代码清单 8-10

```
public static GoodsCategoryList GetChildCategory(string accessToken, string
cate_id = "1")
{
    var obj = new { cate_id = cate_id };
    var url =
        string.Format("https://api.weixin.qq.com/merchant/category/getsub?
        access_token={0}", accessToken);
    return Utils.PostResult<GoodsCategoryList>(obj, url);
}
```

8.2.2 获取指定子分类的所有 SKU

SKU 指的是库存量单位，此处的 SKU 指的是添加商品时，商品的规格列表。如图 8-5 所示，类别为“笔记本电脑”的规则。如果选择的商品没有多规格信息，则调用接口获取 SKU 时会返回 `{"errcode":-1,"errmsg":"system error"}`。

接口地址如下：

`https://api.weixin.qq.com/merchant/category/getsku?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

请求示例：`{"cate_id": 537074292}`。

请求成功后返回的数据示例如代码清单 8-11 所示。

商品规格 ☐ 统一规格 ☒ 多规格

颜色 编辑

<input type="checkbox"/> 白色	<input type="checkbox"/> 宝蓝色	<input type="checkbox"/> 橙红色	<input type="checkbox"/> 橙黄色	<input type="checkbox"/> 橙色	<input type="checkbox"/> 淡黄色
<input type="checkbox"/> 粉红色	<input type="checkbox"/> 粉色	<input type="checkbox"/> 咖啡色	<input type="checkbox"/> 黑色	<input type="checkbox"/> 红色	<input type="checkbox"/> 黄色
<input type="checkbox"/> 花色	<input type="checkbox"/> 灰色	<input type="checkbox"/> 浅灰色	<input type="checkbox"/> 浅蓝色	<input type="checkbox"/> 浅绿色	<input type="checkbox"/> 浅紫色
<input type="checkbox"/> 金色	<input type="checkbox"/> 酒红色	<input type="checkbox"/> 军绿色	<input type="checkbox"/> 蓝色	<input type="checkbox"/> 冷灰色	<input type="checkbox"/> 绿色
<input type="checkbox"/> 抹茶绿色	<input type="checkbox"/> 玫红色	<input type="checkbox"/> 米白色	<input type="checkbox"/> 墨绿色	<input type="checkbox"/> 巧克力色	<input type="checkbox"/> 肉粉色
<input type="checkbox"/> 深灰色	<input type="checkbox"/> 深卡其布色	<input type="checkbox"/> 深蓝色	<input type="checkbox"/> 深紫色	<input type="checkbox"/> 天蓝色	<input type="checkbox"/> 透明
<input type="checkbox"/> 土黄色	<input type="checkbox"/> 驼色	<input type="checkbox"/> 香槟色	<input type="checkbox"/> 象牙白色	<input type="checkbox"/> 杏色	<input type="checkbox"/> 薰衣草紫
<input type="checkbox"/> 荧光	<input type="checkbox"/> 银色	<input type="checkbox"/> 月光银	<input type="checkbox"/> 中黄色	<input type="checkbox"/> 紫红色	<input type="checkbox"/> 紫罗兰色
<input type="checkbox"/> 紫色	<input type="checkbox"/> 棕褐色	<input type="checkbox"/> 棕色	<input type="checkbox"/> 其他		

添加规格

图 8-5

代码清单 8-11

```

{
  "errcode": 0,
  "errmsg": "success",
  "sku_table": [
    {
      "id": "1075741873",
      "name": "颜色",
      "value_list": [
        {
          "id": "1079742375",
          "name": "撞色"
        },
        {
          "id": "1079742376",
          "name": "桔色"
        }
      ]
    }
  ]
}

```

根据上述代码创建实体类 SkuList，如代码清单 8-12 所示。



然后创建类 GoodsCategoryList,

代码清单 8-12

```
using System.Collections.Generic;
namespace WxApi.ReceiveEntity.Shop
{
    /// <summary>
    /// 分类的 SKU 列表
    /// </summary>
    public class SkuList : ErrorEntity
    {
        public List<Sku> sku_table { get; set; }
    }

    public class Sku : BaseEntity
    {
        public List<BaseEntity> value_list { get; set; }
    }
}
```

最后, 在 GoodsManage 类中添加获取 SKU 列表的代码, 如代码清单 8-13 所示。

代码清单 8-13

```
public static SkuList GetsSkuList(string cate_id, string accessToken)
{
    var obj = new { cate_id = cate_id };
    var url =
        string.Format("https://api.weixin.qq.com/merchant/category/getsku?access_token={0}", accessToken);
    return Utils.PostResult<SkuList>(obj, url);
}
```

8.2.3 获取指定分类的所有属性

大部分分类都有特有的属性, 如笔记本电脑有硬盘容量、内存大小、屏幕尺寸等。茶饮料有产地、品牌等属性。在添加商品时也需要指定商品的属性, 所以需要使用接口获取分类的所有属性。请注意, 并不是所有的类别都有特有的属性, 如保险。当类别没有属性时, 调用接口时将返回{"errcode":-1,"errmsg":"system error"}, 开发者不用惊讶, 并不是你的程序错了, 而是此类别没有属性。接口地址如下:



`https://api.weixin.qq.com/merchant/category/getproperty?access_token=ACCESS_TOKEN`

HTTP 请求方式: POST。

请求示例: `{"cate_id": 537074292}`。

调用成功后, 返回的数据示例如代码清单 8-14 所示。

代码清单 8-14

```
{
  "errcode": 0,
  "errmsg": "success",
  "properties": [
    {
      "id": "1075741879",
      "name": "品牌",
      "property_value": [
        {
          "id": "200050867",
          "name": "VIC&#38"
        },
        {
          "id": "200050868",
          "name": "Kate&#38"
        }
      ]
    },
    {
      "id": "123456789",
      "name": "颜色",
      "property_value": ...
    }
  ]
}
```

根据上述数据示例, 创建实体类 `PropertyList`, 如代码清单 8-15 所示。

代码清单 8-15

```
using System.Collections.Generic;
namespace WxApi.ReceiveEntity.Shop
```



```
{
    /// <summary>
    /// 分类属性列表
    /// </summary>
    public class PropertyList : ErrorEntity
    {
        public List<Properties> properties { get; set; }
    }
    public class Properties : BaseEntity
    {
        public List<BaseEntity> property_value { get; set; }
    }
}
```

创建实体后，在 GoodsManage 类中添加获取所有属性的实现方法，如代码清单 8-16 所示。

代码清单 8-16

```
public static PropertyList GetPropertyList(string cate_id, string
accessToken)
{
    var obj = new { cate_id = cate_id };
    var url =
String.Format("https://api.weixin.qq.com/merchant/category/getproperty?
access_token={0}", accessToken);
    return Utils.PostResult<PropertyList>(obj, url);
}
```

8.2.4 增加商品

增加商品的接口地址如下：

https://api.weixin.qq.com/merchant/create?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

由于在创建商品时需要编辑的信息太多，如分类、属性、SKU、价格、快递等，因此调用上述接口增加商品需要 POST 的 JSON 数据也是很复杂的，如图 8-6 所示的是 JSON 的结构。



从图 8-6 中可以看出,商品的实体中应该包括 product_base、sku_list、attrext、delivery_info 四个属性。除了 sku_list 是数组类型外,其他三个属性都是 obj 类型。product_base 的结构如图 8-7 所示。

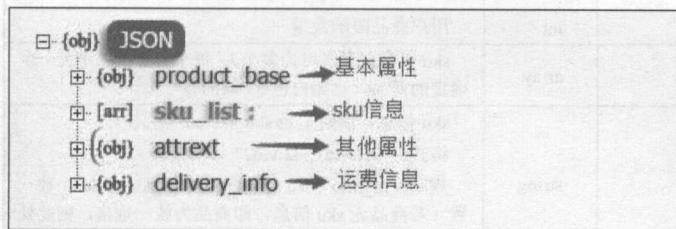


图 8-6

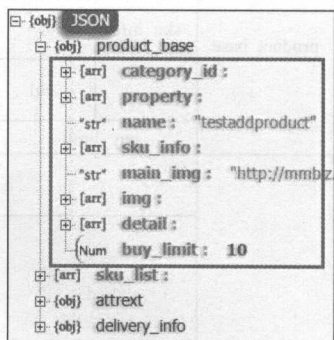


图 8-7

下面将分别讲解每个属性的具体结构,如表 8-1 所示。

表 8-1

字 段		是否必选	类 型	说 明	
product_base			obj	基本属性	
	name	必选	string	商品名称	
	category	必选	array	商品分类 ID，商品分类列表请通过 8.2.1 节中的获取指定分类的所有子分类方法获取。此处填分类的叶子节点即可。虽然类型是数组，但只需要一个元素	
	main_img	必选	string	商品主图（图片需调用本节开头讲到的图片上传接口获得图片 URL；否则无法添加商品。图片分辨率推荐尺寸为 640px×600px）	
	img	必选	array	商品图片列表（图片需调用图片上传接口获得图片 URL 填写至此；否则无法添加商品。图片分辨率推荐尺寸为 640px×600px）	
	detail		必选	array	商品详情列表，显示在客户端的商品详情页内
		text		string	文字描述
		img		string	图片（图片需调用图片上传接口获得图片 URL 填写至此；否则无法添加商品）
	property			array	商品属性列表，属性列表请通过 8.2.3 节中的获取指定分类的所有属性获取
		id		string	属性 id
vid			string	属性值 id	



续表

字 段		是否必选	类 型	说 明
product_base	sku_info		array	商品 SKU 定义, SKU 列表请通过 8.2.2 节中获取指定子分类的所有 SKU 获取
		id	string	sku 属性 (SKU 列表中的 id, 支持自定义 SKU, 格式为 "\$xxx", xxx 即显示在客户端中的字符串)
		vid	array	SKU 值 (SKU 列表中的 vid, 如需自定义 SKU, 格式为 "\$xxx", xxx 即显示在客户端中的字符串)
	buy_limit		int	用户商品限购数量
sku_list			array	sku 信息列表 (可为多个), 每个 sku 信息串为一个确定的商品, 比如白色 37 码的鞋
	sku_id		string	sku 信息, 参照上述 sku_table 的定义 格式: "id1:vid1;id2:vid2" 规则: id_info 的组合个数必须与 sku_table 个数一致 (若商品无 sku 信息, 即商品为统一规格, 则此处赋值为空字符串即可)
	ori_price		int	sku 原价 (单位: 分)
	price		int	sku 微信价 (单位: 分, 微信价必须比原价小, 否则添加商品失败)
	icon_url		string	sku iconurl (图片需调用图片上传接口获得图片 URL)
	quantity		int	sku 库存
	product_code		string	商家商品编码
attrext			obj	商品其他属性
	isPostFree		int	是否包邮 (0——否, 1——是), 如果包邮, delivery_info 字段可省略
	isHasReceipt		int	是否提供发票 (0——否, 1——是)
	isUnderGuaranty		int	是否保修 (0——否, 1——是)
	isSupportReplace		int	是否支持退换货 (0——否, 1——是)
	location		obj	商品所在地地址
		country	string	国家。如中国
		province	string	省份。如湖北省
		city	string	城市。如武汉市
	address		string	详细地址
delivery_info			必选	运费信息
	delivery_type		int	运费类型 (0——使用下面 express 字段的默认模板, 1——使用 template_id 代表的邮费模板, 详见邮费模板相关 API)
	template_id		string	邮费模板 ID
	express	id		快递 ID
		price		运费 (单位: 分)

根据表 8-1 所示的属性结构, 首先创建各个属性的实体类。如代码清单 8-17 所示的是



商品基本属性实体类。

代码清单 8-17

```
/// <summary>
/// sku 信息实体
/// </summary>
public class SkuInfo
{
    public string id { get; set; }
    public List<string> vid { get; set; }
}

/// <summary>
/// 属性键值实体
/// </summary>
public class Property
{
    /// <summary>
    /// 属性 ID
    /// </summary>
    public string id { get; set; }
    /// <summary>
    /// 属性值
    /// </summary>
    public string vid { get; set; }
}

/// <summary>
/// 商品详情
/// </summary>
public class Detail
{
    /// <summary>
    /// 文字描述
    /// </summary>
    public string text { get; set; }
    /// <summary>
    /// 图片（图片需调用图片上传接口获得图片 URL 填写至此；否则无法添加商品）
```



```
    /// </summary>
    public string img { get; set; }
}

/// <summary>
/// 商品基本信息
/// </summary>
public class ProductBase
{
    /// <summary>
    /// 商品名称
    /// </summary>
    public string name { get; set; }
    /// <summary>
    /// 商品分类 id
    /// </summary>
    public List<string> category_id { get; set; }
    /// <summary>
    /// 商品主图（图片需调用图片上传接口获得图片 URL 填写至此；否则无法添加商品。
    /// 图片分辨率推荐尺寸为 640px×600px）
    /// </summary>
    public string main_img { get; set; }
    /// <summary>
    /// 商品图片列表（图片需调用图片上传接口获得图片 URL 填写至此；否则无法添加商品。
    /// 图片分辨率推荐尺寸为 640px×600px）
    /// </summary>
    public List<string> img { get; set; }
    /// <summary>
    /// 商品详情列表，显示在客户端的商品详情页内
    /// </summary>
    public List<Detail> detail { get; set; }
    /// <summary>
    /// 商品属性列表
    /// </summary>
    public List<Property> property { get; set; }
    /// <summary>
    /// 商品 sku 定义。如需自定义 sku，格式为"$xxx"。如 id="$整件库存" vid="$1000"
```



```
    /// </summary>
    public List<SkuInfo> sku_info { get; set; }
    /// <summary>
    /// 用户商品限购数量
    /// </summary>
    public int buy_limit { get; set; }
}
```

商品 SKU 列表中的 sku 实体类如代码清单 8-18 所示。

代码清单 8-18

```
public class ProductSku
{
    /// <summary>
    /// sku 信息 规则: id_info 的组合个数必须与 sku_table 个数一致 (若商品无 sku 信息,
    /// 即商品为统一规格, 则此处赋值为空字符串即可)
    /// </summary>
    public string sku_id { get; set; }
    /// <summary>
    /// sku 原价 (单位 : 分)
    /// </summary>
    public int ori_price { get; set; }
    /// <summary>
    /// sku 微信价 (单位: 分, 微信价必须比原价小; 否则添加商品失败)
    /// </summary>
    public int price { get; set; }
    /// <summary>
    /// sku iconurl (图片需调用图片上传接口获得图片 URL)
    /// </summary>
    public string icon_url { get; set; }
    /// <summary>
    /// sku 库存
    /// </summary>
    public int quantity { get; set; }
    /// <summary>
    /// 商家商品编码
    /// </summary>
    public string product_code { get; set; }
}
```




商品其他属性实体类如代码清单 8-19 所示。

代码清单 8-19

```
/// <summary>
/// 商品所在地地址
/// </summary>
public class Location
{
    /// <summary>
    /// 国家
    /// </summary>
    public string country { get; set; }
    /// <summary>
    /// 省份
    /// </summary>
    public string province { get; set; }
    /// <summary>
    /// 城市
    /// </summary>
    public string city { get; set; }
    /// <summary>
    /// 详细地址
    /// </summary>
    public string address { get; set; }
}

public class AttrExt
{
    /// <summary>
    /// 是否包邮 (0——否, 1——是), 如果包邮, 则商品运费信息 delivery_info 字段可省略
    /// </summary>
    public int isPostFree { get; set; }
    /// <summary>
    /// 是否提供发票 (0——否, 1——是)
    /// </summary>
    public int isHasReceipt { get; set; }
    /// <summary>
    /// 是否保修 (0——否, 1——是)
    /// </summary>
```




```
public int isUnderGuaranty { get; set; }  
/// <summary>  
/// 是否支持退换货 (0——否, 1——是)  
/// </summary>  
public int isSupportReplace { get; set; }  
/// <summary>  
/// 商品所在地地址  
/// </summary>  
public Location Location { get; set; }  
}
```

商品运费信息实体类如代码清单 8-20 所示。

代码清单 8-20

```
/// <summary>  
/// 运费实体  
/// </summary>  
public class express  
{  
    /// <summary>  
    /// 快递 ID  
    /// </summary>  
    public string id { get; set; }  
    /// <summary>  
    /// 运费(单位: 分)  
    /// </summary>  
    public string price { get; set; }  
}  
/// <summary>  
/// 运费信息  
/// </summary>  
public class DeliveryInfo  
{  
    /// <summary>  
    /// 运费类型 0——使用 express 字段的默认模板, 1——使用 template_id 代表的邮费模板  
    /// </summary>  
    public int delivery_type { get; set; }  
    /// <summary>
```



```
/// 邮费模板 ID  
/// </summary>  
public string template_id { get; set; }  
}
```

在上述实体类 `DeliveryInfo` 中的 `template_id` 表示的是邮费模板 ID，此参数需要调用邮费模板相关接口获取，具体实现方式将在 8.3 节中进行讲解。

商品各个属性的类型定义完成后，最后就需要定义商品的实体类了，如代码清单 8-21 所示。

代码清单 8-21

```
public class Product  
{  
    /// <summary>  
    /// 商品基本信息  
    /// </summary>  
    public ProductBase product_base { get; set; }  
    /// <summary>  
    /// sku 列表  
    /// </summary>  
    public List<ProductSku> sku_list { get; set; }  
    /// <summary>  
    /// 其他属性  
    /// </summary>  
    public AttrExt attrext { get; set; }  
    /// <summary>  
    /// 运费信息  
    /// </summary>  
    public DeliveryInfo delivery_info { get; set; }  
}
```

另外，还需要创建请求接口后，返回信息的实体。正常情况下，调用增加商品接口后，返回的信息如代码清单 8-22 所示。

代码清单 8-22

```
{  
    "errcode": 0,
```



```
"errmsg": "success",  
"product_id": "pDF3iYwktviE3BzU3BKisWWi9Nkw"  
}
```

根据上述代码结构，创建实体类 `ProductId`，如代码清单 8-23 所示。

代码清单 8-23

```
namespace WxApi.ReceiveEntity.Shop  
{  
    public class ProductId:ErrorEntity  
    {  
        public string product_id { get; set; }  
    }  
}
```

实体创建完毕后，在 `GoodsManage` 类中添加调用增加商品接口的实现方法，如代码清单 8-24 所示。

代码清单 8-24

```
public static ProductId AddGoods(Product goods, string accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/create?  
access_token={0}", accessToken);  
    return Utils.PostResult<ProductId>(goods, url);  
}
```

8.2.5 修改商品

修改商品的接口地址如下：

`https://api.weixin.qq.com/merchant/update?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

由于是修改商品，因此在请求的时候需要带上目标商品的 ID，其他属性就和增加商品的请求实体一致了，如代码清单 8-25 所示。



代码清单 8-25

```
namespace WxApi.SendEntity.Shop
{
    /// <summary>
    /// 商品的所有信息, 包括商品 ID
    /// </summary>
    public class ProductInfo:Product
    {
        /// <summary>
        /// 商品 ID
        /// </summary>
        public string product_id { get; set; }
        /// <summary>
        /// 商品状态(0—全部, 1—上架, 2—下架)
        /// </summary>
        public int status { get; set; }
    }
}
```

在修改商品时, 从未上架商品的所有信息均可修改; 否则商品的名称 (name)、商品分类 (category)、商品属性 (property) 这 3 个字段不可修改。

修改商品的代码如代码清单 8-26 所示。

代码清单 8-26

```
public static ErrorEntity UpdateGoods(ProductInfo goods, string
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/update?
        access_token={0}", accessToken);
    return Utils.PostResult<ProductId>(goods, url);
}
```

8.2.6 查询商品

查询商品分为两种方式: 一种是根据商品 ID 获取单个商品 (修改商品信息时需要),



另一种是获取指定状态的商品列表。

根据商品 ID 获取商品信息的接口地址如下：

`https://api.weixin.qq.com/merchant/get?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

请求示例：`{"product_id": "pDF3iYwktviE3BzU3BKISWWi9Nkw"}`。

调用成功后，返回的 JSON 数据实体类如代码清单 8-27 所示。

代码清单 8-27

```
using WxApi.SendEntity.Shop;
namespace WxApi.ReceiveEntity.Shop
{
    /// <summary>
    /// 根据商品 ID 获取商品详细信息返回实体
    /// </summary>
    public class ProductRevEntity:ErrorEntity
    {
        /// <summary>
        /// 商品详细信息
        /// </summary>
        public ProductInfo product_info { get; set; }
    }
}
```

实体创建完毕后，在 `GoodsManage` 中添加实现方法，如代码清单 8-28 所示。

代码清单 8-28

```
public static ProductRevEntity GetProductInfo(string productId, string
accessToken)
{
    var url = String.Format("https://api.weixin.qq.com/merchant/get?access_
token={0}", accessToken);
    var obj = new { product_id = productId };
    return Utils.PostResult<ProductRevEntity>(obj, url);
}
```




获取指定状态的商品列表的接口地址如下：

`https://api.weixin.qq.com/merchant/getbystatus?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

请求示例：{"status":0}。

status 表示的是商品状态（0——全部，1——上架，2——下架）

请求成功后，返回的数据实体如代码清单 8-29 所示。

代码清单 8-29

```
using System.Collections.Generic;
using WxApi.SendEntity.Shop;
namespace WxApi.ReceiveEntity.Shop
{
    public class ProductList:ErrorEntity
    {
        public List<ProductInfo> products_info { get; set; }
    }
}
```

实体创建完毕后，在 GoodsManage 中添加获取指定状态的所有商品的实现方法，如代码清单 8-30 所示。

代码清单 8-30

```
public static ProductList GetProductList(string accessToken,int status=0)
{
    var url =
        String.Format("https://api.weixin.qq.com/merchant/getbystatus?
        access_token={0}", accessToken);
    var obj = new { status = status };
    return Utils.PostResult<ProductList>(obj, url);
}
```

8.2.7 删除商品

删除商品的接口地址如下：



`https://api.weixin.qq.com/merchant/del?access_token=ACCESS_TOKEN`

HTTP 请求方式: POST。

请求示例: `{"product_id": "pDF3iYwktviE3BzU3BKISWWi9Nkw"}`。

具体的调用方式如代码清单 8-31 所示。

代码清单 8-31

```
public static ErrorEntity DelGoods(string productId, string accessToken)
{
    var url = string.Format("https://api.weixin.qq.com/merchant/
del?access_token={0}", accessToken);
    var obj = new { product_id = productId };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

8.2.8 商品上下架

商品的上下架是通过 status 属性控制的。接口地址如下:

`https://api.weixin.qq.com/merchant/modproductstatus?access_token=ACCESS_TOKEN`

HTTP 请求方式: POST。

请求示例: `{"product_id": "pDF3iYwktviE3BzU3BKISWWi9Nkw", "status": 0}`。

具体的调用方式如代码清单 8-32 所示。

代码清单 8-32

```
public static ErrorEntity UpdateStatus(string productId, string status,
string accessToken)
{
    var url =
    string.Format("https://api.weixin.qq.com/merchant/modproductstatus?
access_token={0}", accessToken);
    var obj = new { status = status, product_id=productId };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```



8.2.9 修改商品库存

修改库存包括增加与减少，微信官方对这两种操作也分别提供了接口，接口地址如下所示。

增加库存: https://api.weixin.qq.com/merchant/stock/add?access_token=ACCESSTOKEN

减少库存: https://api.weixin.qq.com/merchant/stock/reduce?access_token=ACCESSTOKEN

这两个接口地址是相似的，请求方式都是 POST，请求的示例如代码清单 8-33 所示。

代码清单 8-33

```
{
  "product_id": "pDF3iY5EYkMxs4-tF8xedyES5GQI",
  "sku_info": "10000983:10000995;10001007:10001010",
  "quantity": 20
}
```

在上述代码中，product_id 表示的是商品的 ID，sku_info 表示的是要修改的商品 sku，quantity 表示增加或减少的数量。最终的实现代码如代码清单 8-34 所示。

代码清单 8-34

```
/// <summary>
/// 增加或减少库存
/// </summary>
/// <param name="productId">商品 ID</param>
/// <param name="skuInfo">sku 信息，格式"id1:vid1;id2:vid2"，如商品为统一规格，
/// 则此处赋值为空字符串即可</param>
/// <param name="quantity">增加或减少的库存数量</param>
/// <param name="flag">操作标志。0 减少。1 增加</param>
/// <param name="accessToken"></param>
public static ErrorEntity UpdateStock(string productId, string skuInfo,
string quantity, int flag, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/stock/{0}?
access_token={1}", flag == 1 ? "add" : "reduce", accessToken);
    var obj = new { product_id = productId, sku_info = skuInfo, quantity =
```



```
quantity };  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

8.3 邮费模板管理接口

8.3.1 增加邮费模板

接口地址如下:

https://api.weixin.qq.com/merchant/express/add?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求的 JSON 数据包格式如图 8-8 所示。

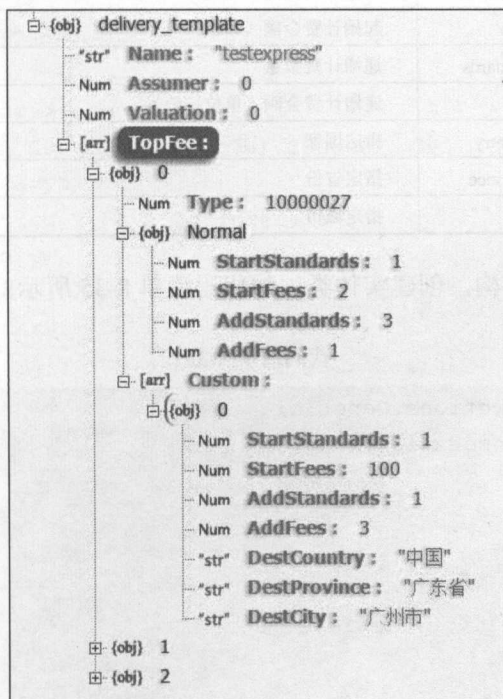


图 8-8



表 8-2 是各个字段的详细说明。

表 8-2

字 段		说 明	
Name		邮费模板名称	
Assumer		支付方式（0——买家承担运费，1——卖家承担运费）	
Valuation		计费单位（0——按件计费，1——按重量计费，2——按体积计费，目前只支持按件计费，默认为 0）	
TopFee	具体运费计算		
	Type	快递类型 ID：平邮（10000027）、快递（10000028）、EMS（10000029）	
	Normal		默认邮费计算方法
		StartStandards	起始计费数量（比如计费单位是按件，填 2 代表起始计费为 2 件）
		StartFees	起始计费金额（单位：分）
		AddStandards	递增计费数量
		AddFees	递增计费金额（单位：分）
		Custom	
	StartStandards		起始计费数量
	StartFees		起始计费金额（单位：分）
	AddStandards		递增计费数量
	AddFees		递增计费金额（单位：分）
	DestCountry		指定国家
	DestProvince		指定省份
	DestCity		指定城市

根据表 8-2 所示的结构, 创建实体类, 如代码清单 8-35 所示。

代码清单 8-35

```
using System.Collections.Generic;
namespace WxApi.SendEntity.Shop
{
    /// <summary>
    /// 运费模板
    /// </summary>
    public class DeliveryTemplate
    {
        /// <summary>
        /// 邮费模板名称
```



```
/// </summary>
public string Name { get; set; }
/// <summary>
/// 支付方式 (0——买家承担运费, 1——卖家承担运费)
/// </summary>
public int Assumer { get; set; }
/// <summary>
/// 计费单位 (0——按件计费, 1——按重量计费, 2——按体积计费, 目前只支持按件计
/// 费, 默认为 0)
/// </summary>
public int Valuation { get; set; }
/// <summary>
/// 具体运费计算
/// </summary>
public List<TopFee> TopFee { get; set; }

}

public class TopFee
{
    /// <summary>
    /// 快递类型 ID: 10000027(平邮) 10000028(快递) 10000029(EMS)
    /// </summary>
    public string Type { get; set; }
    /// <summary>
    /// 默认邮费计算方法
    /// </summary>
    public NormalFormula Normal { get; set; }
    /// <summary>
    /// 指定地区邮费计算方法
    /// </summary>
    public List<CustomFormula> Custom { get; set; }
}

/// <summary>
/// 默认邮费计算方法实体
/// </summary>
public class NormalFormula
{
```



```
    /// <summary>
    /// 起始计费数量
    /// </summary>
    public int StartStandards { get; set; }
    /// <summary>
    /// 起始计费金额（单位：分）
    /// </summary>
    public int StartFees { get; set; }
    /// <summary>
    /// 递增计费数量
    /// </summary>
    public int AddStandards { get; set; }
    /// <summary>
    /// 递增计费金额（单位：分）
    /// </summary>
    public int AddFees { get; set; }
}
/// <summary>
/// 指定地区邮费计算方法实体
/// </summary>
public class CustomFormula:NormalFormula
{
    /// <summary>
    /// 指定国家
    /// </summary>
    public string DestCountry { get; set; }
    /// <summary>
    /// 指定省份
    /// </summary>
    public string DestProvince { get; set; }
    /// <summary>
    /// 指定城市
    /// </summary>
    public string DestCity { get; set; }
}
```

请求接口后，对应的响应信息的实体如下所示：



```
namespace WxApi.ReceiveEntity.Shop
{
    /// <summary>
    /// 增加邮费模板接口返回实体
    /// </summary>
    public class DeliveryIDEntity:ErrorEntity
    {
        public string template_id { get; set; }
    }
}
```

实体创建完毕后,在项目的 Shop 文件夹中添加类 DeliveryManage,用于封装邮费模板相关的接口。在类中添加“增加邮费模板”的方法,如代码清单 8-36 所示。

代码清单 8-36

```
public static DeliveryIDEntity AddTemplate(DeliveryTemplate template,
string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/express/add?access_
token={0}", accessToken);
    var obj = new {delivery_template = template};
    return Utils.PostResult<DeliveryIDEntity>(obj, url);
}
```

8.3.2 删除邮费模板

接口地址如下:

https://api.weixin.qq.com/merchant/express/del?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"template_id": 231321231}。

实现代码如代码清单 8-37 所示。

代码清单 8-37

```
public static ErrorEntity DelTemplate(string templateId, string accessToken)
```




```
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/express/del?access_  
token={0}", accessToken);  
    var obj = new { template_id =templateId};  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

8.3.3 修改邮费模板

接口地址如下:

https://api.weixin.qq.com/merchant/express/update?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"template_id": 123456, "delivery_template": ...}。

其中, delivery_template 表示的是邮费模板信息, 字段详情和 8.3.1 节中增加邮费模板的一致。根据请求示例创建实体, 如代码清单 8-38 所示。

代码清单 8-38

```
namespace WxApi.SendEntity.Shop  
{  
    /// <summary>  
    /// 邮费模板信息实体  
    /// </summary>  
    public class DeliveryTemplateInfo:DeliveryTemplate  
    {  
        public string template_id { get; set; }  
    }  
}
```

请求实体创建完毕后, 修改邮费模板的实现代码如代码清单 8-39 所示。

代码清单 8-39

```
public static ErrorEntity UpdateTemplate(DeliveryTemplateInfo dti,string  
accessToken)
```



```
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/express/update?  
access_token={0}", accessToken);  
    return Utils.PostResult<DeliveryIDEntity>(dti, url);  
}
```

8.3.4 获取邮费模板

和获取商品一样，获取邮费模板也分为两种情况：一种是根据模板 ID 获取单个模板信息，另一种是获取所有模板信息。

根据模板 ID 获取邮费模板的接口地址如下：

https://api.weixin.qq.com/merchant/express/getbyid?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例：{"template_id": 231321231}。

根据请求之后返回的 JSON 数据包创建实体类，如代码清单 8-40 所示。

代码清单 8-40

```
using WxApi.SendEntity.Shop;  
namespace WxApi.ReceiveEntity.Shop  
{  
    /// <summary>  
    /// 运费模板信息  
    /// </summary>  
    public class DeliveryInfoRev:ErrorEntity  
    {  
        public DeliveryTemplateInfo template_info { get; set; }  
    }  
}
```

代码清单 8-41 为根据模板 ID 获取邮费模板的实现代码。

代码清单 8-41

```
public static DeliveryInfoRev GetTemplateInfo(string templateId, string
```



```
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/express/getbyid?
access_token={0}", accessToken);
    var obj = new { template_id = templateId };
    return Utils.PostResult<DeliveryInfoRev>(obj, url);
}
```

获取所有邮费模板的接口地址如下:

https://api.weixin.qq.com/merchant/express/getall?access_token=ACCESS_TOKEN

HTTP 请求方式: GET。

根据返回的 JSON 数据包创建的实体类如代码清单 8-42 所示。

代码清单 8-42

```
using System.Collections.Generic;
using WxApi.SendEntity.Shop;
namespace WxApi.ReceiveEntity.Shop
{
    /// <summary>
    /// 运费模板列表
    /// </summary>
    public class DeliveryList:ErrorEntity
    {
        public List<DeliveryTemplateInfo> templates_info { get; set; }
    }
}
```

代码清单 8-43 为获取所有邮费模板的实现代码。

代码清单 8-43

```
public static DeliveryList GetList(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/express/getall?
access_token={0}", accessToken);
```



```
return Utils.GetResult<DeliveryList>(url);
```

8.4 商品分组管理

8.4.1 增加分组

接口地址如下:

https://api.weixin.qq.com/merchant/group/add?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求数据示例如代码清单 8-44 所示。

代码清单 8-44

```
{
  "group_detail" : {
    "group_name": "测试分组",
    "product_list" : [
      "pDF3iY9cEWyMimNlKbik_NYJTzYU",
      "pDF3iY4kpZagQfwJ_LVQBaoC-LsM"
    ]
  }
}
```

在上述代码中, group_name 表示的是商品分组名称, product_list 表示的是此分组中的商品 ID 列表。请求成功后, 正常情况下将返回如代码清单 8-45 所示的代码。

代码清单 8-45

```
{
  "errcode":0,
  "errmsg":"success",
  "group_id": 19
}
```




根据上述代码创建实体类，如代码清单 8-46 所示。

代码清单 8-46

```
namespace WxApi.ReceiveEntity.Shop
{
    public class GroupId : ErrorEntity
    {
        public string group_id { get; set; }
    }
}
```

实体创建完毕后，在 Shop 文件夹中添加 GroupManage 类，用于封装商品分组相关的方法。代码清单 8-47 所示的是增加商品分组的方法。

代码清单 8-47

```
public static GroupId Add(string groupName, List<string> productList, string
accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/group/add?access_
token={0}", accessToken);
    var obj = new { group_detail = new { group_name = groupName, product_list
= productList } };
    return Utils.PostResult<GroupId>(obj, url);
}
```

8.4.2 删除分组

接口地址如下：

https://api.weixin.qq.com/merchant/group/del?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例：{"group_id": 231321231}。

实现代码如代码清单 8-48 所示。



代码清单 8-48

```
public static ErrorEntity Delete(string groupId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/group/del?
        access_token={0}", accessToken);
    var obj = new { group_id = groupId };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

8.4.3 修改分组名

接口地址如下:

https://api.weixin.qq.com/merchant/group/propertymod?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"group_id": 28, "group_name": "特惠专场"}。

实现代码如代码清单 8-49 所示。

代码清单 8-49

```
public static ErrorEntity UpdateName(string groupId, string groupName,
string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/group/propertymod?
        access_token={0}", accessToken);
    var obj = new { group_id = groupId, group_name = groupName };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

8.4.4 修改分组商品

接口地址如下:

https://api.weixin.qq.com/merchant/group/productmod?access_token=token

HTTP 请求方式: POST。



请求示例如代码清单 8-50 所示。

代码清单 8-50

```
{
  "group_id": 28,
  "product":
  [
    {
      "product_id": "pDF3iY-CgqlAL3k8Ilz-6sj0UYpk", "mod_action": 1
    },
    {
      "product_id": "pDrriY-CrelAL3k8Ilz-7sj0U5Ypk", "mod_action": 0
    }
  ]
}
```

在上述代码中，`mod_action` 表示的是操作类型；0——删除，1——增加。根据示例代码创建实体类，如代码清单 8-51 所示。

代码清单 8-51

```
namespace WxApi.SendEntity.Shop
{
    /// <summary>
    /// 分组商品信息
    /// </summary>
    public class ModProduct
    {
        /// <summary>
        /// 商品 ID
        /// </summary>
        public string product_id { get; set; }
        /// <summary>
        /// 修改操作（0——删除，1——增加）
        /// </summary>
        public int mod_action { get; set; }
    }
}
```



代码清单 8-52 为修改分组商品的具体实现。

HTTP 请求方式: GET。

代码清单 8-52

```
public static ErrorEntity UpdateProduct(string groupId, List<ModProduct>
product, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/group/productmod?
access_token={0}", accessToken);
    var obj = new { group_id = groupId, product = product };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

8.4.5 获取分组信息

获取分组信息分为两种方式:一种是根据分组 ID 获取分组详细信息,另一种是获取所有分组列表。

根据分组 ID 获取分组信息的接口地址如下:

https://api.weixin.qq.com/merchant/group/getbyid?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"group_id": 29}。

正常情况下,请求之后返回的 JSON 如代码清单 8-53 所示。

代码清单 8-53

```
{
    "errcode": 0,
    "errmsg": "success",
    "group_detail": {
        "group_id": 200077549,
        "group_name": "最新上架",
        "product_list": ["pDF3iYzZoY-Budrzt8O6Ixrw", "pDF3iY3pnWSGJcO2
MpS2Nxy3"]
    }
}
```




根据上述代码结构，创建实体类，如代码清单 8-54 所示。

代码清单 8-54

```
using System.Collections.Generic;
namespace WxApi.ReceiveEntity.Shop
{
    public class GroupInfo:ErrorEntity
    {
        public GroupDetail group_detail { get; set; }
    }
    public class GroupDetail
    {
        /// <summary>
        /// 分组 ID
        /// </summary>
        public string group_id { get; set; }
        /// <summary>
        /// 分组名称
        /// </summary>
        public string group_name { get; set; }
        /// <summary>
        /// 商品 ID 列表
        /// </summary>
        public List<string> product_list { get; set; }
    }
}
```

根据分组 ID 获取分组信息的代码如代码清单 8-55 所示。

代码清单 8-55

```
public static GroupInfo GetDetail(string groupId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/group/getbyid?access_token={0}", accessToken);
    var obj = new { group_id = groupId; };
    return Utils.PostResult<GroupInfo>(obj, url);
}
```

获取所有分组列表的接口地址如下：



`https://api.weixin.qq.com/merchant/group/getall?access_token=ACCESS_TOKEN`

HTTP 请求方式: GET。

请求成功后, 返回示例代码如代码清单 8-56 所示。

代码清单 8-56

```
{
  "errcode": 0,
  "errmsg": "success",
  "groups_detail": [
    { group_detail.....}
  ]
}
```

根据上述示例代码的 JSON 结构, 创建实体类, 如代码清单 8-57 所示。

代码清单 8-57

```
using System.Collections.Generic;
namespace WxApi.ReceiveEntity.Shop
{
  /// <summary>
  /// 分组列表实体
  /// </summary>
  public class GroupList : ErrorEntity
  {
    public List<GroupDetail> groups_detail { get; set; }
  }
}
```

代码清单 8-58 为获取所有分组列表的实现代码。

代码清单 8-58

```
public static GroupList GetList(string accessToken)
{
  var url =
    string.Format("https://api.weixin.qq.com/merchant/group/getall?
    access_token={0}", accessToken);
  return Utils.GetResult<GroupList>(url);
}
```



8.5 货架管理

8.5.1 增加货架

通俗地讲，货架就是用来展示商品的。在微信小店中，一个货架可由一个或多个货架控件组成。目前微信官方提供的货架控件有 5 种，开发者可调用接口自由组合。

增加货架的接口地址如下：

`https://api.weixin.qq.com/merchant/shelf/add?access_token=ACCESS_TOKEN`

HTTP 请求方式：POST。

图 8-9 所示的是请求接口时需要 POST 的 JSON 数据的结构。

其中，`module_infos` 表示的是货架控件数组，`shelf_banner` 表示的是货架招牌图片，`shelf_name` 表示的是货架名称。由于请求“增加货架”接口的数据比较复杂，在请求接口前需要创建与请求 JSON 数据相关联的实体映射。而要创建请求的实体映射之前必须搞清楚各个货架控件的结构，因此下面将对各个控件一一进行讲解。

控件 1 由一个分组组成，展示该分组指定数量的商品列表，如图 8-10 所示。



图 8-9



图 8-10



请求时的 JSON 数据包如代码清单 8-59 所示。

代码清单 8-59

```
{
  "group_info": {
    "filter": {
      "count": 4
    },
    "group_id": "4535435435"
  },
  "eid": 1
}
```

在上述代码中，eid 表示的是控件的 ID，控件 1 的 ID 就是 1，控件 2 的 ID 就是 2。
group_info.group_id 表示的是分组的 ID，count 表示的是该控件展示的商品个数。

控件 2 由多个分组组成（最多 4 个分组），展示指定分组的名称，如图 8-11 所示。



8-11

请求时的 JSON 数据包如代码清单 8-60 所示。

代码清单 8-60

```
{
  "group_infos": {
    "groups": [
      {
```




```
        "group_id": "4325435434"
      },
      {
        "group_id": "504353545"
      }
    ]
  },
  "eid": 2
}
```

控件 3 由一个分组组成，展示指定分组的分组图片。开发者可以理解为一个广告位或者 banner，如图 8-12 所示。



8-12

请求时的 JSON 数据包如代码清单 8-61 所示。

代码清单 8-61

```
{
  "group_info": {
    "group_id": "3445345435",
    "img": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl29nqqObBwFwnIX3licVpNfV5Jm64z4I0TTicv0TjN7Vl9bykUUibYKIOjicAwIt6Oy0Y6a1Rjp5Tos8tg/0"
  },
  "eid": 3
}
```



上述代码中的 `img` 值需要调用上传图片接口获取。建议分辨率为 `600px×208px`。

控件 4 由多个分组组成（最多 3 个分组），展示指定分组图片，如图 8-13 所示。

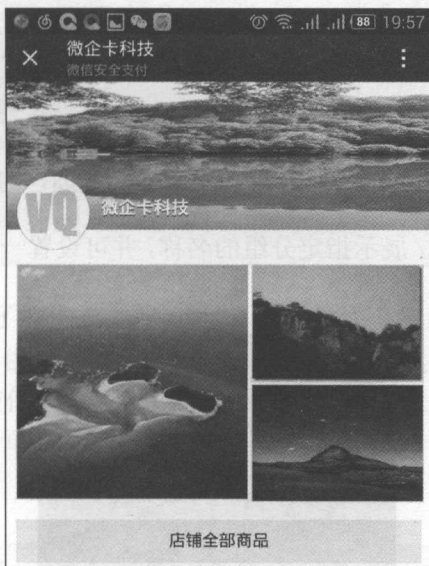


图 8-13

请求时的 JSON 数据包如代码清单 8-62 所示。

代码清单 8-62

```
{
  "group_infos": {
    "groups": [
      {
        "group_id": "3234234243",
        "img": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl29nqqObBwFwnIX3licVPnFV5uUQx7TL"
      }, {
        "group_id": "3232323243",
        "img": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl29nqqObBwFwnIX3licVPnFV5G1kdy3V"
      }, {
        "group_id": "3234234333",
```



```
"img": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl29nqqObBwFwnIX3lic  
VPnFV5uUQx7TL"  
}  
],  
"eid": 4  
}
```

控件 4 中的 3 张图片的尺寸分别推荐为: 350px×350px, 244px×172px, 244px×172px。

控件 5 由多个分组组成, 展示指定分组的名称, 并可设置一个全屏的背景图, 如图 8-14 所示。

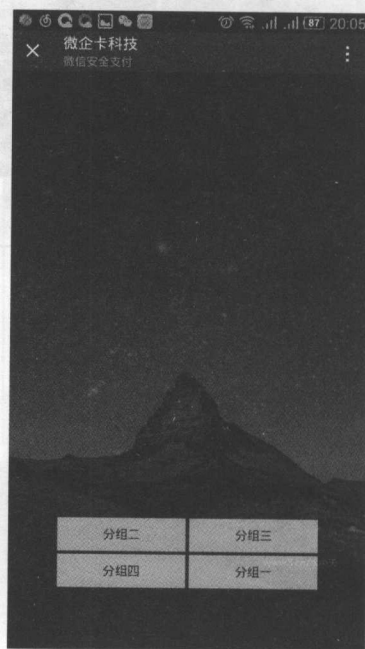


图 8-14

请求时的 JSON 数据包如代码清单 8-63 所示。

代码清单 8-63

```
{  
  "group_infos": {
```



```
"groups": [
  {"group_id": "333534246"},
  {"group_id": "333342546"},
  {"group_id": "333342546"},
  {"group_id": "334323546"},
],
"img_background": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZ129nqqObBwFwn
IX3licVPnFV5uUQx7TLx4tB9qZfbe3JmqR4Nk"
},
"eid": 5
}
```

在上述 5 种控件中,除了控件 5 之外,其他 4 种控件均可随意组合。从各个控件的 JSON 数据可以看出,每个控件都包含属性 `eid`。为了方便调用,创建实体时,可创建一个控件基类,各个控件的实体继承控件基类。由于基类只包含一个属性,并没有实际的意义,所以基类应该是抽象类。代码清单 8-64 为各个控件的实体映射。

代码清单 8-64

```
using System.Collections.Generic;

namespace WxApi.SendEntity.Shop
{
    #region 货架控件基类
    /// <summary>
    /// 货架控件基类
    /// </summary>
    public abstract class ShelfModule
    {
        public int eid { get; set; }
    }
    #endregion
    #region 控件 1
    public class Filter
    {
        public int count { get; set; }
    }

    public class OneGroupInfo
```




```
{
    /// <summary>
    /// 该控件展示商品个数信息
    /// </summary>
    public Filter filter { get; set; }
    public string group_id { get; set; }
}
/// <summary>
/// 控件 1 实体。由一个分组组成，展示该分组指定数量的商品列表，可与除控件 5 之外的控
/// 件共用
/// </summary>
public class ShelfModuleOne : ShelfModule
{
    public ShelfModuleOne() { eid = 1; }
    /// <summary>
    /// 分组信息
    /// </summary>
    public OneGroupInfo group_info { get; set; }
}

#endregion
#region 控件 2
public class GroupIdEntity
{
    public string group_id { get; set; }
}
public class GroupIds
{
    /// <summary>
    /// 分组列表
    /// </summary>
    public List<GroupIdEntity> groups { get; set; }
}
/// <summary>
/// 控件 2 实体。由多个分组组成（最多有 4 个分组），展示指定分组的名称。可与除控件 5
/// 之外的控件共用
/// </summary>
public class ShelfModuleTwo : ShelfModule
```



```
{
    public ShelfModuleTwo() { eid = 2; }
    public GroupIds group_infos { get; set; }
}
#endregion
#region 控件3
public class GroupImg
{
    /// <summary>
    /// 分组 ID
    /// </summary>
    public string group_id { get; set; }
    /// <summary>
    /// 分组照片（图片需调用图片上传接口获得图片 URL 填写至此；否则添加货架失败。
    /// 3 个分组建议分辨率分别为：350px×350px，244px×172px，244px×172px）
    /// </summary>
    public string img { get; set; }
}
/// <summary>
/// 控件3 实体。由一个分组组成，展示指定分组的分组图片。可与除控件5 之外的控件共用
/// </summary>
public class ShelfModuleThree : ShelfModule
{
    public ShelfModuleThree() { eid = 3; }
    public GroupImg group_info { get; set; }
}

#endregion
#region 控件4

public class GroupInfos
{
    public List<GroupImg> groups { get; set; }
}
/// <summary>
/// 控件4 实体。由多个分组组成（最多3 个分组），展示指定分组的分组图片。可与除控件5
/// 之外的控件共用
/// </summary>
```



```
public class ShelfModuleFour : ShelfModule
{
    public ShelfModuleFour() { eid = 4; }
    public GroupInfos group_infos { get; set; }
}
#endregion
#region 控件 5
public class FiveGroupInfo
{
    public List<GroupIdEntity> groups { get; set; }
    public string img_background { get; set; }
}
/// <summary>
/// 控件 5 实体。由多个分组组成，展示指定分组的名称，不可与其他控件联合使用
/// </summary>
public class ShelfModuleFive : ShelfModule
{
    public ShelfModuleFive() { eid = 5; }
    public FiveGroupInfo group_infos { get; set; }
}
#endregion
}
```

货架控件实体创建完成后，根据增加货架时请求的 JSON 数据创建货架实体类 Shelf，如代码清单 8-65 所示。

代码清单 8-65

```
namespace WxApi.SendEntity.Shop
{
    /// <summary>
    /// 货架实体
    /// </summary>
    public class Shelf
    {
        /// <summary>
        /// 货架 ID。增加货架时，此参数无效
        /// </summary>
    }
}
```



```
public int shelf_id { get; set; }  
/// <summary>  
/// 货架信息  
/// </summary>  
/// public ShelfData shelf_data { get; set; }  
/// <summary>  
/// 货架招牌图片 URL (图片需调用图片上传接口获得图片 URL 填写至此; 否则添加货架  
/// 失败。建议尺寸为 640px×120px。仅控件 1~4 有 banner, 控件 5 没有 banner)  
/// </summary>  
public string shelf_banner { get; set; }  
/// <summary>  
/// 货架名  
/// </summary>  
public string shelf_name { get; set; }  
}  
}
```

上述代码中的属性 `shelf_id` 为货架 ID, 在调用增加货架接口时, 此属性无效。此属性是为了兼容修改货架接口及其他需要使用到货架 ID 的接口。 `shelf_data` 的类型为 `ShelfData`, 如代码清单 8-66 所示。

代码清单 8-66

```
using System.Collections.Generic;  
namespace WxApi.SendEntity.Shop  
{  
    public class ShelfData  
    {  
        /// <summary>  
        /// 货架信息  
        /// </summary>  
        public List<ShelfModule> module_infos { get; set; }  
    }  
}
```

在请求增加货架接口后, 返回的信息对应的实体类如下所示:

```
namespace WxApi.ReceiveEntity.Shop  
{
```




```
/// <summary>
/// 增加货架时, 返回的实体
/// </summary>
public class ShelfId:ErrorEntity
{
    /// <summary>
    /// 货架 ID
    /// </summary>
    public string shelf_id { get; set; }
}
```

请求实体创建完毕后, 调用“增加货架”接口的代码就简单多了。为了调用方便, 在项目的 Shop 文件中添加类 ShelfManage, 用于封装货架相关接口操作。类中的方法 Add 为增加货架的实现代码, 如代码清单 8-67 所示。

代码清单 8-67

```
public static ShelfId Add(Shelf shelf,string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/shelf/add?access_
token={0}", accessToken);
    return Utils.PostResult<ShelfId>(shelf, url);
}
```

8.5.2 删除货架

接口地址如下:

https://api.weixin.qq.com/merchant/shelf/del?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"shelf_id": 1}。

实现代码如代码清单 8-68 所示。



代码清单 8-68

```
public static ErrorEntity Delete(string shelfId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/shelf/del?
access_token={0}", accessToken);
    var obj = new { shelf_id =shelfId};
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

8.5.3 修改货架

接口地址如下:

https://api.weixin.qq.com/merchant/shelf/mod?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

修改货架 POST 的 JSON 数据包和增加货架时基本一致。由于需要知道修改的是哪一个货架,因此在调用时需要将货架的 ID 传入。考虑到兼容性,在增加货架的请求实体中已经加入了货架 ID,所以,请求修改货架的实体和增加货架的实体是同一个实体类。只是在调用增加货架时,货架 ID 是无效的。代码清单 8-69 就是修改货架的实现代码。

代码清单 8-69

```
public static ErrorEntity Update(Shelf shelf, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/shelf/mod?access_
token={0}", accessToken);
    return Utils.PostResult<ErrorEntity>(shelf, url);
}
```

8.5.4 获取货架信息

获取货架信息包括根据货架 ID 获取单个货架信息和获取所有货架信息。根据货架 ID 获取货架信息的接口地址如下:



https://api.weixin.qq.com/merchant/shelf/getbyid?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例: {"shelf_id": 1}。

请求成功后, 返回的数据示例如代码清单 8-70 所示。

代码清单 8-70

```
{
  "errcode": 0,
  "errmsg": "success",
  "shelf_info": {"module_infos": [...]},
  "shelf_banner": "http://mmbiz.qpic.cn/mmbiz/4whpV1VZl2ibp2DgDXiaic6W
df1MpNdInS8qUia2Bzt1PulgPl",
  "shelf_name": "新建货架",
  "shelf_id": 97
}
```

根据上述示例代码创建实体类 ShelfInfo, 如代码清单 8-71 所示。

代码清单 8-71

```
using WxApi.SendEntity.Shop;
namespace WxApi.ReceiveEntity.Shop
{
    public class ShelfInfo : ErrorEntity
    {
        public ShelfData shelf_info { get; set; }
        public int shelf_id { get; set; }
        public string shelf_banner { get; set; }
        public string shelf_name { get; set; }
    }
}
```

根据货架 ID 获取货架信息的代码如代码清单 8-72 所示。

代码清单 8-72

```
public static ShelfInfo GetInfo(int shelfId, string accessToken)
```



```
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/shelf/getbyid?
access_token={0}", accessToken);
    var res = Utils.HttpPost(url, "{\"shelf_id\":\"+shelfId+\"}");
    //将返回的 JSON 数据反序列化为实体对象
    var tempe = JsonConvert.DeserializeObject<ShelfInfo>(res);
    //由于在实体对象中包含控件列表，且控件列表是以基类的形式展现的。在反序列化的过程
    //中，并不会判断控件的子类类型，所以需要单独反序列化控件列表
    var jobj = JsonConvert.DeserializeObject<JObject>(res);
    var module_infos = jobj["shelf_info"]["module_infos"];
    tempe.shelf_info.module_infos.Clear();
    foreach (JToken info in module_infos)
    {
        switch (info.Value<int>("eid"))
        {
            case 1: tempe.shelf_info.module_infos.Add(info.ToObject<
                ShelfModuleOne>()); break;
            case 2: tempe.shelf_info.module_infos.Add(info.ToObject<
                ShelfModuleTwo>()); break;
            case 3: tempe.shelf_info.module_infos.Add(info.ToObject<
                ShelfModuleThree>()); break;
            case 4: tempe.shelf_info.module_infos.Add(info.ToObject<
                ShelfModuleFour>()); break;
            case 5: tempe.shelf_info.module_infos.Add(info.ToObject<
                ShelfModuleFive>()); break;
        }
    }
    return tempe;
}
```

获取所有货架列表的接口地址如下：

https://api.weixin.qq.com/merchant/shelf/getall?access_token=ACCESS_TOKEN

HTTP 请求方式：GET。

请求成功后，返回示例代码如代码清单 8-73 所示。



代码清单 8-73

```
{  
    "errcode": 0,  
    "errmsg": "success",  
    "shelves": [  
        .....货架信息列表  
    ]  
}
```

根据上述示例代码的 JSON 结构，创建实体类，如代码清单 8-74 所示。

代码清单 8-74

```
using System.Collections.Generic;  
using WxApi.SendEntity.Shop;  
  
namespace WxApi.ReceiveEntity.Shop  
{  
    /// <summary>  
    /// 货架列表  
    /// </summary>  
    public class Shelves:ErrorEntity  
    {  
        public List<Shelf> shelves { get; set; }  
    }  
}
```

代码清单 8-75 所示的就是获取所有分组列表的实现代码。

代码清单 8-75

```
public static Shelves GetAllList(string accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/shelf/getall?  
access_token={0}", accessToken);  
    return Utils.GetResult<Shelves>(url);  
}
```



8.5.5 自定义货架页面

微信小店的货架支持开放给开发者使用，即开发者可以将自己的页面作为货架，通过 JavaScript API 来调起微信客户端原生的商品详情页。

开发者需要预先通过调用增加商品接口上传商品获取 `product_id` 后，才能在自己的页面通过 JavaScript API 来调起商品详情页。需要注意的是，即使开发者将自己的页面作为货架，但由于商品存储仍在微信服务器，因此用户下单后，订单、库存管理等事务，开发者仍需要根据微信小店系列接口来完成。

具体 JavaScript API 调用示例代码如代码清单 8-76 所示。

代码清单 8-76

```
<script>
function openProductView() {
    alert(WeixinJSBridge);
    if (typeof WeixinJSBridge == "undefined")
        return false;

    var pid = "pxb7Qsl0KYZKvcK0XgDQVZaDgTMQ"; //只需要传递
    WeixinJSBridge.invoke('openProductViewWithPid', {
        "pid": pid
    }, function (res) {
        // 返回 res.err_msg, 取值
        // open_product_view_with_id:ok 打开成功
        alert(res.err_msg);
        if (res.err_msg != "open_product_view_with_id:ok") {
            WeixinJSBridge.invoke('openProductView', {
                "productInfo": "{\"product_id\":\"" + pid + "\", \"product_type\"
":0}"
            }, function (res) {
                alert(res.err_msg);
            });
        }
    });
}
</script>
```



8.6 订单管理

面页梁武义宝自 8.2.8

8.6.1 订单付款通知

用户在微信中付款成功后，微信服务器会将订单付款通知推送到开发者在公众平台网站中设置的回调 URL（在开发模式中设置）中。如未设置回调 URL，则获取不到该事件推送。事件推送的内容如代码清单 8-77 所示。

代码清单 8-77

```
<xml>
  <ToUserName><![CDATA[weixin_medial]]></ToUserName>
  <FromUserName><![CDATA[oDF3iYyVlek46AyTBbMRVV8VZV1I]]></FromUserName>
  <CreateTime>1398144192</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[merchant_order]]></Event>
  <OrderId><![CDATA[test_order_id]]></OrderId>
  <OrderStatus>2</OrderStatus>
  <ProductId><![CDATA[test_product_id]]></ProductId>
  <SkuInfo><![CDATA[10001:1000012;10002:100021]]></SkuInfo>
</xml>
```

根据上述 XML 代码，创建实体类 OrderEventMsg，如代码清单 8-78 所示。

代码清单 8-78

```
namespace WxApi.MsgEntity
{
    /// <summary>
    /// 微信小店订单事件实体
    /// </summary>
    public class OrderEventMsg : EventMsg
    {
        /// <summary>
        /// 订单 ID
        /// </summary>
        public string OrderId { get; set; }
        /// <summary>
```



```

    /// 订单状态
    /// </summary>
    public string OrderStatus { get; set; }
    /// <summary>
    /// 商品 ID
    /// </summary>
    public string ProductId { get; set; }
    /// <summary>
    /// SKU 信息
    /// </summary>
    public string SkuInfo { get; set; }
}

```

然后在事件的枚举 EventType 中添加项 MERCHANT_ORDER, 如图 8-15 所示。

之后就是在 MsgFactory 类中, 处理事件相关的代码中添加映射订单实体的代码, 如图 8-16 所示。



图 8-15

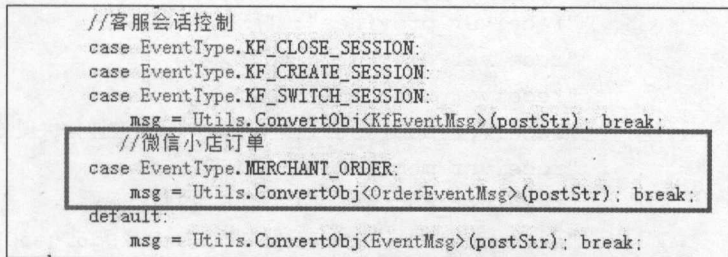


图 8-16

最后, 在 WxTest 项目的 wx.ashx 中绑定微信订单相关的回调。可参考多客服或自定义菜单相关章节中关于处理事件的方法。在此不一一赘述了。

8.6.2 根据订单 ID 获取订单详情

在获取到订单付款通知后, 开发者可根据获取到的订单 ID 调用接口获取订单详情。接口地址如下:

https://api.weixin.qq.com/merchant/order/getbyid?access_token=ACCESS_TOKEN



HTTP 请求方式: POST。

请求示例: {"order_id": 1}。

请求成功后, 返回的数据示例如代码清单 8-79 所示。

代码清单 8-79

```
{
  "errcode": 0,
  "errmsg": "success",
  "order": {
    "order_id": "7197417460812533543",
    "order_status": 6,
    "order_total_price": 6,
    "order_create_time": 1394635817,
    "order_express_price": 5,
    "buyer_openid": "oDF3iY17NsDAW4UP2qzJXPsz1S9Q",
    "buyer_nick": "likeacat",
    "receiver_name": "张小猫",
    "receiver_province": "广东省",
    "receiver_city": "广州市",
    "receiver_zone": "天河区",
    "receiver_address": "华景路一号南方通信大厦 5 楼",
    "receiver_mobile": "123456789",
    "receiver_phone": "123456789",
    "product_id": "pDF3iYx7KDQVGzB7kDg6Tge5OKFo",
    "product_name": "某某某产品",
    "product_price": 1,
    "product_sku": "10000983:10000995;10001007:10001010",
    "product_count": 1,
    "product_img": "http://img2.paipaiimg.com/00000000/item-52B87243-63CCF66C00000000040100003565C1EA.0.300x300.jpg",
    "delivery_id": "1900659372473",
    "delivery_company": "059Yunda",
    "trans_id": "1900000109201404103172199813"
  }
}
```



根据上述 JSON 数据创建实体类 OrderRev，如代码清单 8-80 所示。

代码清单 8-80

```
namespace WxApi.ReceiveEntity.Shop
{
    public class OrderRev:ErrorEntity
    {
        /// <summary>
        /// 订单信息
        /// </summary>
        public OrderInfo order { get; set; }
    }

    public class OrderInfo
    {
        /// <summary>
        /// 订单 ID
        /// </summary>
        public string order_id { get; set; }
        /// <summary>
        /// 订单状态
        /// </summary>
        public string order_status { get; set; }
        /// <summary>
        /// 订单总价格（单位：分）
        /// </summary>
        public string order_total_price { get; set; }
        /// <summary>
        /// 订单创建时间
        /// </summary>
        public string order_create_time { get; set; }
        /// <summary>
        /// 订单运费价格（单位：分）
        /// </summary>
        public string order_express_price { get; set; }
        /// <summary>
        /// 买家微信 openid
        /// </summary>
```



```
public string buyer_openid { get; set; }  
/// <summary>  
/// 买家微信昵称  
/// </summary>  
public string buyer_nick { get; set; }  
/// <summary>  
/// 收货人姓名  
/// </summary>  
public string receiver_name { get; set; }  
/// <summary>  
/// 收货地址省份  
/// </summary>  
public string receiver_province { get; set; }  
/// <summary>  
/// 收货地址城市  
/// </summary>  
public string receiver_city { get; set; }  
/// <summary>  
/// 收货地址区/县  
/// </summary>  
public string receiver_zone { get; set; }  
/// <summary>  
/// 收件人详细地址  
/// </summary>  
public string receiver_address { get; set; }  
/// <summary>  
/// 收件人手机号  
/// </summary>  
public string receiver_mobile { get; set; }  
/// <summary>  
/// 收件人固定电话  
/// </summary>  
public string receiver_phone { get; set; }  
/// <summary>  
/// 商品 ID  
/// </summary>  
public string product_id { get; set; }  
/// <summary>
```



```
/// 商品名称
/// </summary>
public string product_name { get; set; }
/// <summary>
/// 商品价格 (单位: 分)
/// </summary>
public string product_price { get; set; }
/// <summary>
/// 商品 SKU
/// </summary>
public string product_sku { get; set; }
/// <summary>
/// 商品个数
/// </summary>
public string product_count { get; set; }
/// <summary>
/// 商品图片
/// </summary>
public string product_img { get; set; }
/// <summary>
/// 运单 ID
/// </summary>
public string delivery_id { get; set; }
/// <summary>
/// 物流公司编码
/// </summary>
public string delivery_company { get; set; }
/// <summary>
/// 交易 ID
/// </summary>
public string trans_id { get; set; }
}
}
```

创建实体完毕后, 在项目的 Shop 文件夹中创建 OrderManage 类, 用于封装订单相关的操作。然后在类中添加“根据订单 ID 获取订单详情”的实现代码, 如代码清单 8-81 所示。

代码清单 8-81

```
public static OrderRev GetOrder(string orderId, string accessToken)
```




```
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/order/getbyid?access_token={0}", accessToken);  
    var obj = new {order_id = orderId};  
    return Utils.PostResult<OrderRev>(obj, url);  
}
```

8.6.3 根据订单状态/创建时间获取订单列表

此接口主要是在做订单筛选时使用。接口地址如下：

https://api.weixin.qq.com/merchant/order/getbyfilter?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。

请求示例：{"status": 2, "begintime": 1397130460, "endtime": 1397130470}。

在请求示例中，status 表示的是订单状态（不带该字段——全部状态，2——待发货，3——已发货，5——已完成，8——维权中）。begintime 的格式是 UNIX 时间戳，表示的是订单创建时间的起始时间（不带该字段则不按照时间做筛选）；endtime 表示的是订单创建的终止时间。

接口请求成功后，正常情况下，将返回如代码清单 8-82 所示的 JSON 数据示例。

代码清单 8-82

```
{  
    "errcode": 0,  
    "errmsg": "success",  
    "order_list": [  
        .....订单详情列表  
    ]  
}
```

根据上述 JSON 示例，创建实体类 OrderList，如代码清单 8-83 所示。

代码清单 8-83

```
using System.Collections.Generic;  
namespace WxApi.ReceiveEntity.Shop
```



```
{
    public class OrderList:ErrorEntity
    {
        public List<OrderInfo> order_list { get; set; }
    }
}
```

代码清单 8-84 所示的就是“根据订单状态/创建时间获取订单列表”的实现代码。

代码清单 8-84

```
public static OrderList GetOrderList(string accessToken, int? status=null,
DateTime? beginTime = null,DateTime? endTime = null)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/order/getbyfilter?
access_token={0}", accessToken);
    object obj = null;
    if (beginTime != null && endTime != null)
    {
        obj = new
        {
            status = status,
            begintime = Utils.ConvertDateTimeInt(beginTime.Value),
            endtime = Utils.ConvertDateTimeInt(endTime.Value)
        };
    }
    else
    {
        obj = new { status = status };
    }
    return Utils.PostResult<OrderList>(obj, url);
}
```

8.6.4 设置订单发货信息

设置订单发货信息的接口地址如下：



https://api.weixin.qq.com/merchant/order/setdelivery?access_token=ACCESS_TOKEN

HTTP 请求方式: POST。

请求示例如代码清单 8-85 所示。

代码清单 8-85

```
{
  "order_id": "7197417460812533543",
  "delivery_company": "059Yunda",
  "delivery_track_no": "1900659372473",
  "need_delivery": 1,
  "is_others": 0
}
```

在上述代码中, 各个字段的说明如表 8-3 所示。

表 8-3

字 段	说 明
order_id	订单 ID
delivery_company	物流公司 ID (参考表 8-4 的“物流公司 ID”; 当 need_delivery 为 0 时, 可不填本字段; 当 need_delivery 为 1 时, 该字段不能为空; 当 need_delivery 为 1 且 is_others 为 1 时, 本字段填写其他物流公司名称)
delivery_track_no	运单 ID(当 need_delivery 为 0 时, 可不填本字段; 当 need_delivery 为 1 时, 该字段不能为空;)
need_delivery	商品是否需要物流 (0——不需要, 1——需要。若无该字段, 则默认为需要物流)
is_others	是否为表 8-4 之外的其他物流公司 (0——否, 1——是。若无该字段, 则默认为不是其他物流公司)

表 8-4

物 流 公 司	ID	物 流 公 司	ID
邮政 EMS	Fsearch_code	顺丰速运	003shunfeng
申通快递	002shentong	韵达快运	059Yunda
中通速递	066zhongtong	宅急送	064zhaijisong
圆通速递	056yuantong	汇通快运	020huitong
天天快递	042tiantian	易迅快递	zj001yixun



为了方便后期调用,可参考 3.1.1 节(全局返回码与接口频率限制)中,处理全局返回码的方式。

首先将表 8-3 赋值到文本文件中,并命名为 `DeliveryCode`,然后将此文件提交到 Code 资源文件中,如图 8-17 所示。

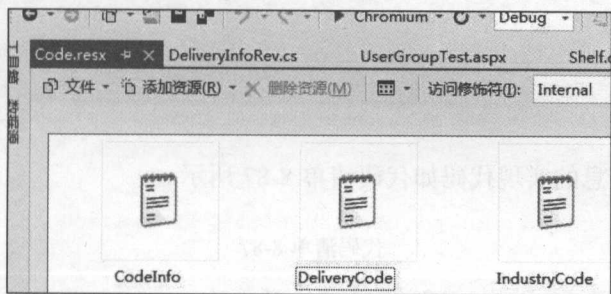


图 8-17

最后添加类 `DeliveryEntity`,用于存储资源文件中的内容,如代码清单 8-86 所示。

代码清单 8-86

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// 快递公司列表
    /// </summary>
    public class DeliveryEntity
    {
        private static Dictionary<string, string> _deliverylist;
        public static Dictionary<string, string> Deliverylist
        {
            get
            {
                //判断值是否为空,如果不为空,则直接返回;否则从资源文件中读取
                if (_deliverylist != null && _deliverylist.Count > 0)
                    return _deliverylist;
                _deliverylist = new Dictionary<string, string>();
                var temp = Code.DeliveryCode.Split(new char[] { '\r', '\n' },
```




```
StringSplitOptions.RemoveEmptyEntries);  
foreach (var strArr in temp.Select(str => str.Split('\t')))  
{  
    _deliverylist.Add(strArr[0], strArr[1]);  
}  
return _deliverylist;  
}  
}  
}
```

设置订单发货信息的实现代码如代码清单 8-87 所示。

代码清单 8-87

```
public static ErrorEntity SetDelivery(string accessToken, string orderId,  
string deliveryCompany,  
string deliveryTrackNo, int needDelivery = 1, int isOthers = 0)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/merchant/order/setdelivery?  
access_token={0}", accessToken);  
    var obj = new  
    {  
        order_id = orderId,  
        delivery_company = deliveryCompany,  
        delivery_track_no = deliveryTrackNo,  
        need_delivery = needDelivery,  
        is_others = isOthers  
    };  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

8.6.5 关闭订单

接口地址如下：

https://api.weixin.qq.com/merchant/order/close?access_token=ACCESS_TOKEN

HTTP 请求方式：POST。



请求示例: {"order_id": 231321231}。

实现代码如代码清单 8-88 所示。

代码清单 8-88

```
public static ErrorEntity CloseOrder(string orderId, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/merchant/order/close?
        access_token={0}", accessToken);
    var obj = new { order_id = orderId};
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

第 9 章 卡券功能接口

9.1 功能简介

微信卡券功能是腾讯为商户提供的一套完整的电子券解决方案，商户可在法律允许的范围内通过该功能实现电子卡券生成、下发、领取、核销的闭环，并使用对账、卡券管理等配套功能。

微信卡券功能可分为 API 接口功能和公众平台卡券功能，使用这两种功能均可实现卡券生成、下发、领取、核销。API 接口功能正是本章要介绍的重点内容。

目前支持优惠券（代金券、折扣券、礼品券、团购券）、会员卡、景点门票、电影票、飞机票、会议门票等多种卡券类型。由于微信 6.0 版本后才支持卡券功能模块，因此低版本用户调用 JSAPI 无效。故此，建议开发者通过 UA 来确定用户当前的版本号后再调用至添加卡包 JSAPI 接口。判断微信版本号的方法已经在第 7 章中讲述了，在此不再赘述。

9.2 开发流程

获取接口权限后，商户开发者需按照以下步骤开发和调试。

（1）创建卡券前的准备

调用上传 LOGO 接口将商户图标上传至微信服务器并获取 logo_url，在用户创建卡券



时使用。

调用门店管理接口获取门店 ID，在用户创建卡券时填入使用门店。

调用获取颜色列表接口，拉取目前支持的卡券颜色值。

(2) 创建卡券

调用创建卡券接口，设置卡券相应信息，获取 cardid，并标注可领取的库存。创建成功后该卡券会自动提交审核，审核结果将通过事件通知告知商户。开发者可调用设置白名单接口设置用户白名单，领取未通过审核的卡券，测试整个卡券的使用流程。

(3) 投放卡券

开发者可生成卡券领取二维码，也可在网页内调用 JavaScript 接口，引导用户领取卡券。调用 JavaScript 接口需通过获取 api_ticket 接口获取 api_ticket，用于签名。

(4) 核销卡券

调用核销接口可对指定卡券进行核销。支持网页内调用 JavaScript 接口拉取卡券列表，用户选择卡券后即可完成核销。

(5) 管理卡券

开发者可对已创建的卡券进行查询、删除、更改、设置失效等操作。同时，在卡券通过审核、卡券被领取、卡券被删除时，均会推送事件通知开发者。

9.3 创建卡券前的准备

9.3.1 上传 LOGO 接口

开发者需调用该接口上传商户图标至微信服务器，获取响应 logo_url，用于卡券创建。上传图片的文件大小限制为 1MB，像素为 300px×300px，支持 JPG 格式。接口地址如下：

https://api.weixin.qq.com/cgi-bin/media/uploading?access_token=ACCESS_TOKEN



HTTP 请求方式: POST/FORM。

请求成功后, 返回的数据如下所示:

```
{"url": "http://mmbiz.qpic.cn/mmbiz/iaL1LJM1mF9aRKPZJkmG8xXhiaHqkKSVMWeN3hLut7X7hicFNjakmxibMLGWpXrEXB33367o7zHN0CwngnQY7zb7g/0"}
```

根据返回的数据创建接口响应的实体类 LogoUrlEntity, 如代码清单 9-1 所示。

代码清单 9-1

```
namespace WxApi.ReceiveEntity
{
    public class LogoUrlEntity:ErrorEntity
    {
        public string url { get; set; }
    }
}
```

在 WxApi 项目中创建类 CardVoucher, 用于封装卡券相关的操作。最后在类中添加上传 LOGO 的实现代码, 如代码清单 9-2 所示。

代码清单 9-2

```
public static LogoUrlEntity UploadLogo(string filePath, string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/cgi-bin/media/uploadimg?access_token={0}", accessToken);
    var formlist = new List<FormEntity>();
    formlist.Add(new FormEntity { IsFile = true, Name = "buffer", Value = filePath });
    return Utils.PostFormResult<LogoUrlEntity>(formlist, url);
}
```

9.3.2 门店管理接口

门店管理接口为商户提供门店批量导入、查询、修改、删除等主要功能, 方便商户快速、高效进行门店管理和操作。下面将对门店管理的各个接口分别进行讲解。



1. 创建门店

创建门店接口是为商户提供创建自己门店数据的接口。门店数据字段越完整，商户页面展示越丰富，就越能吸引用户，并提供曝光度。

调用接口新建门店时所使用的图片 URL 必须为微信自己域名的 URL，因此需要先用上传图片接口上传图片并获取 URL，再创建门店。上传的图片限制文件大小为 1MB，支持 JPG 格式。接口地址如下：

`https://file.api.weixin.qq.com/cgi-bin/media/uploading?access_token=token`

HTTP 请求方式：POST/FORM。

POST 数据格式：buffer。

接口请求成功后，返回的示例如下：

```
{"url": "http://mmbiz.qpic.cn/XXXXX"}
```

根据上述示例创建实体类 `PicUrl`，如代码清单 9-3 所示。

代码清单 9-3

```
namespace WxApi.ReceiveEntity
{
    public class PicUrl : ErrorEntity
    {
        public string url { get; set; }
    }
}
```

在代码实现的过程中，首先在项目中新建 `Store` 类，用于封装门店相关的方法。然后在代码中添加上传图片的方法 `UploadPic`，如代码清单 9-4 所示。

代码清单 9-4

```
public static PicUrl UploadPic(string filePath, string accessToken)
{
    var url = string.Format("https://file.api.weixin.qq.com/cgi-bin/media/uploading?access_token={0}", accessToken);
```



```
var formlist = new List<FormEntity>();  
formlist.Add(new FormEntity { IsFile = true, Name = "buffer", Value =  
filePath });  
return Utils.PostResult<PicUrl>(formlist, url);  
}
```

创建门店接口调用成功后会返回 `errcode`、`errmsg`，但不会实时返回 `poi_id`。成功创建后，门店信息会经过审核，审核通过后方可使用，并获取 `poi_id`。该 `poi_id` 为门店的唯一 `poi_id`，强烈建议自行存储该 `poi_id`，并为后续调用使用。创建门店的接口地址如下：

http://api.weixin.qq.com/cgi-bin/poi/addpoi?access_token=token

HTTP 请求方式：POST。

请求参数的 JSON 格式如图 9-1 所示。

```
{obj} business  
  {obj} base_info  
    "str" sid: "33788392"  
    "str" business_name: "麦当劳"  
    "str" branch_name: "艺苑路店"  
    "str" province: "广东省"  
    "str" city: "广州市"  
    "str" district: "海珠区"  
    "str" address: "艺苑路 11 号"  
    "str" telephone: "020-12345678"  
    [arr] categories:  
      "str" "美食,快餐小吃"  
    Num offset_type: 1  
    Num longitude: 115.32375  
    Num latitude: 25.097486  
    [arr] photo_list:  
      "str" recommend: "麦辣鸡腿堡套餐, 麦乐鸡, 全家桶"  
      "str" special: "免费 wifi, 外卖服务"  
      "str" introduction: "麦当劳是全球大型跨国连锁餐"  
      "str" open_time: "8:00-20:00"  
    Num avg_price: 35
```

图 9-1

详细参数列表说明如表 9-1 所示。



表 9-1

字 段	说 明	是 否 必 填
sid	商户自己的 id, 用于后续审核通过收到 poi_id 的通知时, 做对应关系。请商户自己保证唯一识别性	是
business_name	门店名称. 仅为商户名, 如: 国美. 不应包含地区、店号等信息。错误示例: 北京国美	是
branch_name	分店名称 (不应包含地区信息、不应与门店名重复。错误示例: 北京王府井店)	否
province	门店所在的省份 (直辖市填城市名, 如: 北京市)	是
city	门店所在的城市	是
district	门店所在地区	否
address	门店所在的详细街道地址 (不要填写省市信息)	是
telephone	门店的电话 (纯数字, 区号、分机号均由 “-” 隔开)	是
categories	门店的类型。不同分类用 “,” 隔开, 如: 美食, 川菜, 火锅	是
offset_type	坐标类型, 1 为火星坐标 (目前只能选 1)	是
longitude	门店所在地理位置的经度	是
latitude	门店所在地理位置的纬度	是
photo_list	图片列表, URL 形式, 可以有 multiple 张图片, 尺寸为 640px×340px	是
recommend	推荐品, 餐厅可为推荐菜; 酒店为推荐套房; 景点为推荐游玩景点等, 可填写针对自己行业的推荐内容	否
special	特色服务, 如免费 WiFi、免费停车、送货上门等商户能提供的特色功能或服务	是
introduction	商户简介, 主要介绍商户信息等	否
open_time	营业时间, 24 小时制表示, 用 “-” 连接, 如 8:00-20:00	是
avg_price	人均价格, 大于 0 的整数	否

根据表 9-1 创建请求的实体类 BaseStoreInfo, 如代码清单 9-5 所示。

代码清单 9-5

```
using System.Collections.Generic;

namespace WxApi.SendEntity
{
    public class BaseStoreInfo
    {
        /// <summary>
        /// 商户自定义 id
        /// </summary>
        public string sid { get; set; }
        /// <summary>
        /// 门店名称. 仅为商户名, 如: 国美. 不应包含地区、店号等信息。错误示例: 北京国美
        /// </summary>
```




```
public string business_name { get; set; }
/// <summary>
/// 分店名称（不应包含地区信息、不应与门店名重复。错误示例：北京王府井店）
/// </summary>
public string branch_name { get; set; }
/// <summary>
/// 门店所在的省份（直辖市填城市名，如：北京市）
/// </summary>
public string province { get; set; }
/// <summary>
/// 门店所在的城市
/// </summary>
public string city { get; set; }
/// <summary>
/// 门店所在地区
/// </summary>
public string district { get; set; }
/// <summary>
/// 门店所在的详细街道地址（不要填写省市信息）
/// </summary>
public string address { get; set; }
/// <summary>
/// 门店的电话（纯数字，区号、分机号均由“-”隔开）
/// </summary>
public string telephone { get; set; }
/// <summary>
/// 门店的类型，不同级分类用“,”隔开，如：美食,川菜,火锅
/// </summary>
public string categories { get; set; }
/// <summary>
/// 坐标类型，1 为火星坐标（目前只能选 1）
/// </summary>
public int offset_type { get; set; }
/// <summary>
/// 门店所在地理位置的经度
/// </summary>
public string longitude { get; set; }
/// <summary>
/// 门店所在地理位置的纬度
/// </summary>
public string latitude { get; set; }
/// <summary>
/// 图片列表，URL 形式，可以有多张图片，尺寸为 640px×340px
```



```
/// </summary>
/// public List<PhotoUrl> photo_list { get; set; }
/// <summary>
/// 推荐品, 餐厅可为推荐菜; 酒店为推荐套房; 景点为推荐游玩景点等, 针对自己行业
/// 的推荐内容
/// </summary>
public string recommend { get; set; }
/// <summary>
/// 特色服务, 如免费 WiFi、免费停车、送货上门等商户能提供的特色功能或服务
/// </summary>
public string special { get; set; }
/// <summary>
/// 商户简介, 主要介绍商户信息等
/// </summary>
public string introduction { get; set; }
/// <summary>
/// 营业时间, 24 小时制表示, 用“-”连接, 如 8:00-20:00
/// </summary>
public string open_time { get; set; }
/// <summary>
/// 人均价格, 大于 0 的整数
/// </summary>
public int avg_price { get; set; }
}
public class PhotoUrl
{
    public string photo_url { get; set; }
}
```

实体创建完毕后, 在 Store 中添加实现此接口的方法 Add, 如代码清单 9-6 所示。

代码清单 9-6

```
public static ErrorEntity Add(BaseStoreInfo info, string accessToken)
{
    var url = string.Format("http://api.weixin.qq.com/cgi-bin/poi/
addpoi?access_token={0}", accessToken);
    var obj = new { business = new { base_info = info } };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

新创建的门店在审核通过后, 会以事件形式推送给开发者在微信后台“开发者中心”



设置的回调 URL。推送的 XML 数据包示例如代码清单 9-7 所示。

代码清单 9-7

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1408622107</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[poi_check_notify]]></Event>
  <UniqId><![CDATA[123adb]]></UniqId>
  <PoiId><![CDATA[123123]]></PoiId>
  <Result><![CDATA[fail]]></Result>
  <Msg><![CDATA[xxxxxx]]></Msg>
</xml>
```

在上述示例代码中，UniqID 表示的是商户自己内部的 ID，即创建门店时提交的 sid 参数；PoiId 是微信的门店 ID，微信内门店唯一 ID；Result 表示的是审核结果，成功为 succ 或失败为 fail；Msg 表示的是成功的通知信息，或审核失败的驳回理由。

处理事件消息的方式在 3.3 节中着重讲述过，在此不再赘述。

2. 门店类别列表

在添加门店的实体中有个参数 categories，表示的是门店的类型。添加的时候需要从类别列表中选择，且此参数是必填的，如图 9-2 所示。

但是，微信官方却没有提供类目列表的接口或数据。那应该怎么办呢？因为在微信公众号的后台里添加门店时是可以看到每个分类和子分类的，所以分类列表的数据肯定会在添加店铺时某个 HTTP 请求的响应中。功夫不负有心人，笔者最终通过浏览器的开发者工具在一个 common_template_helper25676c.js 请求中找到了分类列表的数据，如图 9-3 所示。

从图 9-3 可以看出，列表的数据是以 JSON 的方式表示的，所以可以将此段 JSON 数据保存为文本文件 StoreCategory，然后添加到项目的 Code 资源文件中，之后创建实体类 StoreCategory，如代码清单 9-8 所示。

代码清单 9-8

```
using System.Collections.Generic;
```



```
namespace WxApi.ReceiveEntity
```

```
{
```

```
    public class StoreCategory
```

```
    {
```

```
        public string name { get; set; }
```

```
        public string value { get; set; }
```

```
        /// <summary>
```

```
        /// 子列表
```

```
        /// </summary>
```

```
        public List<StoreCategory> sub { get; set; }
```

```
    }
```

```
}
```

图 9-2

	×	Headers	Preview	Response	T
639	}				
640	var n=[{				
641	name:"美食",				
642	value:"",				
643	sub:[{				
644	name:"江浙菜",				
645	value:"",				
646	},{				
647	name:"川菜",				
648	value:"",				
649	},{				
650	name:"川菜",				
651	value:"",				
652	},{				
653	name:"湘菜",				
654	value:"",				
655	},{				
656	name:"东北菜",				

图 9-3

最后在 Store 类中添加如下代码调用 Code 资源文件中的 StoreCategory, 如代码清单 9-9 所示。

代码清单 9-9

```
public static List<StoreCategory> GetCategory()
```

```
{
```

```
    return JsonConvert.DeserializeObject<List<StoreCategory>>
```




```
(Code.StoreCategory);
```

商户在使用门店管理接口时需注意以下几个问题:

- 门店信息全部需要经过审核方能生效, 门店创建完成后, 只会返回创建成功提示, 并不能获得 `poi_id`, 只有经过审核后才能获取 `poi_id`, 收到微信推送的审核结果通知, 并用于微信各个业务场景中。
- 为保证在审核通过后, 获取到的 `poi_id` 能与商户自身数据做对应, 将会允许商户在创建时提交自己内部或自定义的 `sid` (字符串格式, 微信不会对唯一性进行校验, 请商户自己保证), 用于后续获取 `poi_id` 后对数据进行对应。
- 门店的可用状态 `available_state`, 将标记门店相应的审核状态。只有审核通过状态, 才能进行更新, 更新字段仅限扩展字段。
- 扩展字段属于公共编辑信息, 提交更新后将由微信进行审核采纳, 但扩展字段更新并不影响门店的可用状态 (即 `available_state` 仍为审核通过)。当 `update_status` 状态变为 1, 更新中状态时, 不可再次对门店进行更新 (直到微信审核采纳后)。
- 在 `update_status` 为 1, 更新中状态下的门店, 此时调用 `getpoi` 接口, 获取到的扩展字段为更新的最新字段, 但并不是最终结果, 仍需等待微信编辑对扩展字段的建议进行采纳后, 最终决定是否生效 (有可能更新字段不被采纳)。

9.3.3 获取颜色列表接口

此接口用于获取卡券的最新颜色列表, 用于卡券创建。接口地址如下:

https://api.weixin.qq.com/card/getcolors?access_token=TOKEN

HTTP 请求方式: GET/POST。

接口调用成功后, 返回的数据格式如图 9-4 所示。

从图 9-4 中可以看出, `colors` 是个列表, 其中 `name` 表示创建卡券时可以填入的 `color` 名称, `value` 是对应的颜色数值。

根据图 9-4 的 JSON 数据结构创建实体类 `CardColors`, 如代码清单 9-10 所示。

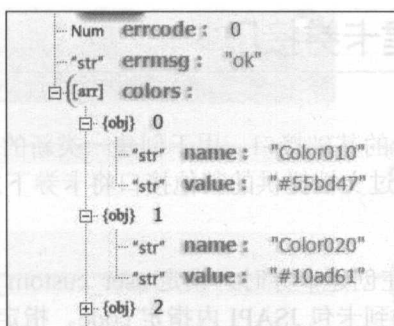


图 9-4

代码清单 9-10

```

using System.Collections.Generic;
namespace WxApi.ReceiveEntity
{
    public class CardColors:ErrorEntity
    {
        public List<Color> colors { get; set; }
    }
    public class Color
    {
        public string name { get; set; }
        public string value { get; set; }
    }
}

```

实体创建完毕后，在 CardVoucher 类中添加获取卡券颜色的方法 GetCardColors，如代码清单 9-11 所示。

代码清单 9-11

```

public static CardColors GetCardColors(string accessToken)
{
    var url =
        string.Format("https://api.weixin.qq.com/card/getcolors?access_
token={0}", accessToken);
    return Utils.GetResult<CardColors>(url);
}

```



9.4 CreateCard 创建卡券接口

创建卡券接口是微信卡券的基础接口，用于创建一类新的卡券，获取 card_id。创建成功并通过审核后，商家可以通过文档提供的其他接口将卡券下发给用户，每次成功领取后，库存数量相应扣除。

需指定 code 的商家必须在创建卡券时，设定 user_custom_code 为 true，且在后续用户领取卡券时，在二维码或添加到卡包 JSAPI 内指定 code。指定 openid 同理。

接口地址如下：

https://api.weixin.qq.com/card/create?access_token=access_token

HTTP 请求方式：POST。

POST 参数字段如图 9-5 和图 9-6 所示。



图 9-5

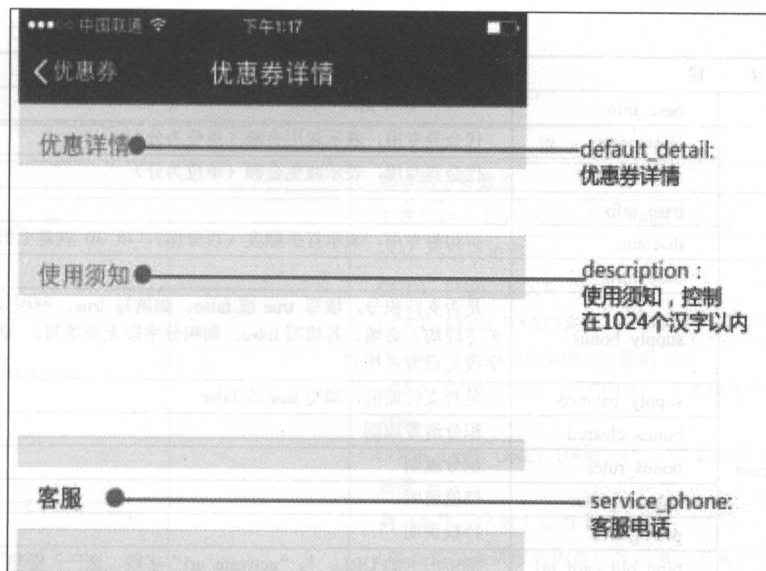


图 9-6

请求的参数列表如表 9-2 所示。

表 9-2

字 段			说 明	是 否 必 填	
card	card_type		卡券信息部分	是	
			卡券类型如下。 通用券：GENERAL_COUPON 团购券：GROUPON 折扣券：DISCOUNT 礼品券：GIFT 代金券：CASH 会员卡：MEMBER_CARD 景点门票：SCENIC_TICKET 电影票：MOVIE_TICKET 飞机票：BOARDING_PASS 红包：LUCKY_MONEY 会议门票：MEETING_TICKET	是	
		general_coupon	base_info	基本的卡券数据，见表 9-3，所有卡券通用	是
			default_detail	描述文本	是
		groupon	base_info		是
			deal_detail	团购券专用，团购详情	是
		gift	base_info		是
			gift	礼品券专用，表示礼品名字	是



续表

字 段			说 明	是 否 必 填
card	cash	base_info		是
		least_cost	代金券专用, 表示起用金额 (单位为分)	否
		reduce_cost	代金券专用, 表示减免金额 (单位为分)	是
	discount	base_info		是
		discount	折扣券专用, 表示打折额度 (百分比)。填 30 就是七折	是
	member_card	base_info		是
		supply_bonus	是否支持积分, 填写 true 或 false。如填写 true, 则积分相关字段均为必填。若填写 false, 则积分字段无须填写。储值字段处理方式相同	是
		supply_balance	是否支持储值, 填写 true 或 false	是
		bonus_cleared	积分清零规则	否
		bonus_rules	积分规则	否
		balance_rules	储值说明	否
		prerogative	特权说明	是
		bind_old_card_url	绑定旧卡的 URL; 与“activate_url”字段二选一, 必填	否
		activate_url	激活会员卡的 URL; 与“bind_old_card_url”字段二选一, 必填	否
		need_push_on_view	为用户单击进入会员卡时是否推送事件	否
	scenic_ticket	base_info		是
		ticket_class	票类型, 例如平日全票、套票等	否
		guide_url	导览图 url	否
	movie_ticket	base_info		是
		detail	电影票详情	否
	boarding_pass	base_info		是
		from	起点, 上限为 18 个汉字	是
		to	终点, 上限为 18 个汉字	是
		flight	航班	是
		departure_time	起飞时间。UNIX 时间戳格式	否
		landing_time	降落时间。UNIX 时间戳格式	否
		check_in_url	在线值机的链接	否
		gate	登机口。如发生登机口变更, 建议商家实时调用该接口变更	否
		boarding_time	登机时间, 只显示“时分”, 不显示日期, 按时间戳格式填写。如发生登机时间变更, 建议商家实时调用该接口变更	否
	lucky_money	air_model	机型, 上限为 8 个汉字	否
		base_info	基本的卡券数据, 见表 9-3, 所有卡券通用。红包类型为非现金红包, 不支持提现	是
	meeting_ticket	base_info		是
		map_url	会场导览图	否
		meeting_detail	会议详情	是

表 9-3 是 baseinfo 字段的详细描述。

表 9-3

字 段		说 明	是 否 必 填
base_info		基本的卡券数据	是
	logo_url	卡券的商户 LOGO，尺寸为 300px×300px	是
	code_type	code 码展示类型如下。 CODE_TYPE_TEXT：文本 CODE_TYPE_BARCODE：一维码 CODE_TYPE_QRCODE：二维码 CODE_TYPE_ONLY_QRCODE：二维码无 code 显示 CODE_TYPE_ONLY_BARCODE：一维码无 code 显示	是
	brand_name	商户名字，字数上限为 12 个汉字 (填写直接提供服务的商户名，第三方商户名填写在 source 字段)	是
	title	券名，字数上限为 9 个汉字(建议涵盖卡券属性、服务及金额)	是
	sub_title		否
	color		是
	notice		是
	description		是
	date_info	使用日期，有效期的信息	是
		type 使用时间的类型，仅支持选择一种时间类型的字段填入。1：固定日期区间，2：固定时长（自领取后按天算）	是
		begin_timestamp type 为 1 时专用，表示起用时间。从 1970 年 1 月 1 日 00:00:00 至起用时间的秒数，最终需转换为字符串形态传入（单位为秒）	是
		end_timestamp type 为 1 时专用，表示结束时间（单位为秒）格式和 begin_timestamp 一样	是
		fixed_term type 为 2 时专用，表示自领取后多少天内有效(单位为天)。领取后若当天有效，则填写 0	是
		fixed_begin_term type 为 2 时专用，表示自领取后多少天开始生效(单位为天)	是
	sku	商品信息	是
		quantity 卡券库存的数量（不支持填写 0 或无限大）	是
	location_id_list	门店位置 ID。商户需在 mp 平台上录入门店信息或调用批量导入门店信息接口获取门店位置 ID	否



续表

字 段		说 明	是 否 必 填
base_info	use_custom_code	是否自定义 code 码。填写 true 或 false，不填代表默认为 false。该字段需单独申请权限	否
	bind_openid	是否指定用户领取，填写 true 或 false。不填代表默认为否	否
	can_share	领取卡券原生页面是否可分享，填写 true 或 false。true 代表可分享。默认为 true	否
	can_give_friend	卡券是否可转赠，填写 true 或 false。true 代表可转赠。默认为 true	否
	get_limit	每人最大领取次数，不填写则默认等于 quantity	否
	service_phone	客服电话	否
	source	第三方来源名，例如同程旅游、格瓦拉	否
	custom_url_name	商户自定义入口名称，与 custom_url 字段共同使用，长度限制在 5 个汉字内	否
	custom_url	商户自定义入口跳转外链的地址链接，跳转页面内容需与自定义 cell 名称保持匹配	否
	custom_url_sub_title	显示在入口右侧的 tips，长度限制在 6 个汉字内	否
	promotion_url_name	营销场景的自定义入口	否
	promotion_url	入口跳转外链的地址链接	否
	promotion_url_sub_title	显示在入口右侧的 tips，长度限制在 6 个汉字内	否

自定义入口显示规则如图 9-7 所示。

优惠券详情

查看公众号

适用门店

立即使用

再次购买

custom_url_name: 第一个自定义入口，卡券激活前（会员卡激活/绑定）和使用后，均会隐藏显示。建议填入使用场景的文字，例如：立即使用、在线预约等

promotion_url_name: 第二个自定义入口，卡券激活前和使用后，均会显示，建议填入营销场景的文字，例如尊享服务再次购买等。

图 9-7

在代码实现的过程中，首先根据上面的表 9-2 和表 9-3 创建实体映射。代码清单 9-12 是卡券基本信息的实体映射。



代码清单 9-12

```
using System.Collections.Generic;

namespace WxApi.SendEntity.Card
{
    public class Sku
    {
        /// <summary>
        /// 卡券库存的数量（不支持填写 0 或无限大）
        /// </summary>
        public int quantity { get; set; }
    }

    /// <summary>
    /// 卡券日期信息
    /// </summary>
    public class DateInfo
    {
        /// <summary>
        /// 使用时间的类型，仅支持选择一种时间类型的字段填入。1：固定日期区间，2：固定
        /// 时长（自领取后按天算）
        /// </summary>
        public int type { get; set; }

        /// <summary>
        /// type 为 1 时专用，表示起用时间。UNIX 时间戳
        /// </summary>
        public string begin_timestamp { get; set; }

        /// <summary>
        /// type 为 1 时专用，表示结束时间。UNIX 时间戳
        /// </summary>
        public string end_timestamp { get; set; }

        /// <summary>
        /// type 为 2 时专用，表示自领取后多少天内有效（单位为天）。领取后若当天有效，
        /// 则填写 0
        /// </summary>
        public int? fixed_term { get; set; }

        /// <summary>
```




```
/// type 为 2 时专用，表示自领取后多少天开始生效（单位为天）。若领取后当天生效，
/// 则填写 0
/// </summary>
public int? fixed_begin_term { get; set; }
}
/// <summary>
/// code 码展示类型
/// </summary>
public enum CodeType
{
    /// <summary>
    /// 文本
    /// </summary>
    CODE_TYPE_TEXT,
    /// <summary>
    /// 一维码
    /// </summary>
    CODE_TYPE_BARCODE,
    /// <summary>
    /// 二维码
    /// </summary>
    CODE_TYPE_QRCODE,
    /// <summary>
    /// 二维码无 code 显示
    /// </summary>
    CODE_TYPE_ONLY_QRCODE,
    /// <summary>
    /// 一维码无 code 显示
    /// </summary>
    CODE_TYPE_ONLY_BARCODE
}
}
/// <summary>
/// 卡券基本信息
/// </summary>
public class BaseInfo
{
```



```
/// <summary>
/// 卡券的商户 LOGO, 尺寸为 300px×300px
/// </summary>
public string logo_url { get; set; }
/// <summary>
/// code 码展示类型
/// </summary>
public CodeType code_type { get; set; }
/// <summary>
/// 商户名字, 字数上限为 12 个汉字 (填写直接提供服务的商户名, 第三方商户名填写
/// 在 source 字段)
/// </summary>
public string brand_name { get; set; }
/// <summary>
/// 券名, 字数上限为 9 个汉字 (建议涵盖卡券属性、服务及金额)
/// </summary>
public string title { get; set; }
/// <summary>
/// 券名的副标题, 字数上限为 18 个汉字
/// </summary>
public string sub_title { get; set; }
/// <summary>
/// 券颜色
/// </summary>
public string color { get; set; }
/// <summary>
/// 使用提醒, 字数上限为 12 个汉字 (一句话描述, 展示在首页, 示例: 请出示二维码
/// 核销卡券)
/// </summary>
public string notice { get; set; }
/// <summary>
/// 使用说明. 长文本描述, 可以分行, 上限为 1000 个汉字
/// </summary>
public string description { get; set; }
/// <summary>
/// 使用日期, 有效期的信息
/// </summary>
```



```
public DateInfo date_info { get; set; }
/// <summary>
/// 门店位置 ID。商户需在 mp 平台上录入门店信息或调用批量导入门店信息接口获取
/// 门店位置 ID
/// </summary>
public List<string> location_id_list { get; set; }
/// <summary>
/// 是否自定义 code 码。填写 true 或 false。不填则代表默认为 false。该字段
/// 需单独申请权限
/// </summary>
public bool use_custom_code { get; set; }
/// <summary>
/// 是否指定用户领取。默认为否
/// </summary>
public bool bind_openid { get; set; }
/// <summary>
/// 领取卡券原生页面是否可分享。默认为 true
/// </summary>
public bool can_share { get; set; }
/// <summary>
/// 是否可转增。默认为 true
/// </summary>
public bool can_give_friend { get; set; }
/// <summary>
/// 每人最大领取次数。若不填写，则默认等于库存
/// </summary>
public int get_limit { get; set; }
/// <summary>
/// 客服电话
/// </summary>
public string service_phone { get; set; }
/// <summary>
/// 第三方来源名，例如同程旅游、格瓦拉
/// </summary>
public string source { get; set; }
/// <summary>
/// 商户自定义入口名称，与 custom_url 字段共同使用，长度限制在 5 个汉字内
```




```
/// </summary>
public string custom_url_name { get; set; }
/// <summary>
/// 商户自定义入口跳转外链的地址链接, 跳转页面内容需与自定义 cell 名称保持匹配
/// </summary>
public string custom_url { get; set; }
/// <summary>
/// 显示在入口右侧的 tips, 长度限制在 6 个汉字内
/// </summary>
public string custom_url_sub_title { get; set; }
/// <summary>
/// 营销场景的自定义入口
/// </summary>
public string promotion_url_name { get; set; }
/// <summary>
/// 入口跳转外链的地址链接
/// </summary>
public string promotion_url { get; set; }
/// <summary>
/// 显示在入口右侧的 tips, 长度限制在 6 个汉字内
/// </summary>
public string promotion_url_sub_title { get; set; }

public Sku sku { get; set; }
}
```

每一种类型的卡券都有相同的属性和独有的属性, 所以可以将相同的属性抽象出来, 创建抽象类 BaseCard, 如代码清单 9-13 所示。

代码清单 9-13

```
namespace WxApi.SendEntity.Card
{
    /// <summary>
    /// 卡券基础信息类
    /// </summary>
    public abstract class BaseCard
    {
```




```
        public BaseInfo base_info { get; set; }  
    }  
}
```

下面是每一种类型对应的实体类，每个实体类都继承基类 BaseCard（见代码清单 9-14 到代码清单 9-24）。

代码清单 9-14（飞机票）

```
namespace WxApi.SendEntity.Card  
{  
    public class Boarding_Pass : BaseCard  
    {  
        /// <summary>  
        /// 起点  
        /// </summary>  
        public string from { get; set; }  
        /// <summary>  
        /// 终点  
        /// </summary>  
        public string to { get; set; }  
        /// <summary>  
        /// 航班  
        /// </summary>  
        public string flight { get; set; }  
        /// <summary>  
        /// 起飞时间。UNIX 时间戳格式  
        /// </summary>  
        public string departure_time { get; set; }  
        /// <summary>  
        /// 降落时间。UNIX 时间戳格式  
        /// </summary>  
        public string landing_time { get; set; }  
        /// <summary>  
        /// 在线值机的链接  
        /// </summary>  
        public string check_in_url { get; set; }  
        /// <summary>
```



```
/// 登机口
/// </summary>
public string gate { get; set; }
/// <summary>
/// 登机时间，只显示“时分”，不显示日期。按时间戳格式填写。如发生登记时间变更，
/// 建议商家实时调用
/// </summary>本接口变更
public string boarding_time{ get; set; }
/// <summary>
/// 机型。不超过8个汉字
/// </summary>
public string air_model { get; set; }
}
}
```

代码清单 9-15 (代金券)

```
namespace WxApi.SendEntity.Card
{
    public class Cash : BaseCard
    {
        /// <summary>
        /// 代金券专用，表示起用金额（单位为分）
        /// </summary>
        public int? least_cost { get; set; }
        /// <summary>
        /// 代金券专用，表示减免金额（单位为分）
        /// </summary>
        public int reduce_cost { get; set; }
    }
}
```

代码清单 9-16 (折扣券)

```
namespace WxApi.SendEntity.Card
{
    public class Discount : BaseCard
    {
```



```
/// <summary>
/// 表示打折额度（百分比）。填 30 就是七折
/// </summary>
public int discount { get; set; }
}
}
```

代码清单 9-17（通用卡券）

```
namespace WxApi.SendEntity.Card
{
    public class General_Coupon: BaseCard
    {
        /// <summary>
        /// 描述文本
        /// </summary>
        public string default_detail { get; set; }
    }
}
```

代码清单 9-18（礼品券）

```
namespace WxApi.SendEntity.Card
{
    public class Gift : BaseCard
    {
        /// <summary>
        /// 礼品券专用，表示礼品名字
        /// </summary>
        public string gift { get; set; }
    }
}
```

代码清单 9-19（团购券）

```
namespace WxApi.SendEntity.Card
{
    public class Groupon : BaseCard
    {
        /// <summary>
```



```
/// 团购详情
/// </summary>
public string deal_detail { get; set; }
```

```
}
```

```
}
```

代码清单 9-20 (非现金红包)

```
namespace WxApi.SendEntity.Card
{
    public class Lucky_Money : BaseCard
    {

    }
}
```

代码清单 9-21 (会议门票)

```
namespace WxApi.SendEntity.Card
{
    public class Meeting_Ticket : BaseCard
    {
        /// <summary>
        /// 会议详情
        /// </summary>
        public string meeting_detail { get; set; }
        /// <summary>
        /// 会场导览图
        /// </summary>
        public string map_url { get; set; }
    }
}
```

代码清单 9-22 (会员卡)

```
namespace WxApi.SendEntity.Card
{
    public class Member_Card: BaseCard
    {
        /// <summary>
```




```
/// 是否支持积分。若填写 true，则积分相关字段均为必填。若填写 false，则积分字  
/// 段无须填写。储值字段处理方式相同  
/// </summary>  
public bool supply_bonus { get; set; }  
/// <summary>  
/// 是否支持储值，填写 true 或 false  
/// </summary>  
public bool supply_balance { get; set; }  
/// <summary>  
/// 积分清零规则  
/// </summary>  
public string bonus_cleared { get; set; }  
/// <summary>  
/// 积分规则  
/// </summary>  
public string bonus_rules { get; set; }  
/// <summary>  
/// 储值说明  
/// </summary>  
public string balance_rules { get; set; }  
/// <summary>  
/// 特权说明  
/// </summary>  
public string prerogative { get; set; }  
/// <summary>  
/// 绑定旧卡的 URL，与“activate_url”字段二选一，必填  
/// </summary>  
public string bind_old_card_url { get; set; }  
/// <summary>  
/// 激活会员卡卡的 URL，与“bind_old_card_url”字段二选一，必填  
/// </summary>  
public string activate_url { get; set; }  
/// <summary>  
/// 用户单击进入会员卡时是否推送事件  
/// </summary>  
public bool need_push_on_view { get; set; }  
}  
}
```



代码清单 9-23 (电影票)

```
namespace WxApi.SendEntity.Card
{
    public class Movie_Ticket : BaseCard
    {
        /// <summary>
        /// detail
        /// </summary>
        public string detail { get; set; }
    }
}
```

代码清单 (景点门票) 9-24

```
namespace WxApi.SendEntity.Card
{
    public class Scenic_Ticket : BaseCard
    {
        /// <summary>
        /// 票类型, 例如平日全票、套票等
        /// </summary>
        public string ticket_class { get; set; }
        /// <summary>
        /// 导览图 url
        /// </summary>
        public string guide_url { get; set; }
    }
}
```

接口调用完成后, 返回的数据示例如下所示:

```
{"errcode":0,"errmsg":"ok","card_id":"p1Pj9jr90_SQRaVqYI239KalerkI"}
```

根据上述示例代码创建实体类 CardIdInfo, 如代码清单 9-25 所示。

代码清单 9-25

```
namespace WxApi.ReceiveEntity
{
    public class CardIdInfo:ErrorEntity
```



```
{  
    /// <summary>  
    /// 卡券 ID  
    /// </summary>  
    public string card_id { get; set; }  
}
```

实体创建完毕后,就是接口调用的代码实现了。在 LogoUrlEntity 类中添加方法 Create,如代码清单 9-26 所示。

代码清单 9-26

```
public static CardIdInfo Create(BaseCard info, string accessToken)  
{  
    var cardType = info.GetType().Name;  
    var url = string.Format("https://api.weixin.qq.com/card/create?access_token={0}", accessToken);  
    var dictionary = new Dictionary<string, object>();  
    dictionary.Add("card_type", cardType.ToUpper());  
    dictionary.Add(cardType.ToLower(), info);  
    var oo = new { card = dictionary };  
    return Utils.PostResult<CardIdInfo>(oo, url);  
}
```

在上述代码中,info 参数的类型是抽象类 BaseCard。在调用的时候,根据生成的卡券类型传对应的实体对象。在程序执行的过程中会判断真实的卡券类型。

代码清单 9-27 就是生成会员卡的示例代码。

代码清单 9-27

```
var info = new Member_Card  
{  
    base_info = new BaseInfo  
    {  
        bind_openid = false,  
        brand_name = "微企卡科技",  
        can_give_friend = false,  
        can_share = false,
```



```
color = "Color010",
custom_url = "http://www.baidu.com",
code_type = CodeType.CODE_TYPE_TEXT,
custom_url_name = "查看详情",
custom_url_sub_title = "查看更多优惠",
sku = new Sku { quantity = 50000000 },
date_info = new DateInfo
{
    type = 2,
    fixed_term = 2
},
description = "微企卡科技会员",
get_limit = 3,
logo_url = "http://mmbiz.qpic.cn/mmbiz/icIxxjVX9TaZe6jxWAQQ5ydy
079LGEfQYAvAzk41N4VKKFmkD6P86ZvGFM3SmYlK0SNUAx9FS5GTicLfh3vfpm7A/0",
notice = "请出示二维码享受会员优惠",
promotion_url = "http://qq.com",
promotion_url_sub_title = "点击进入",
promotion_url_name = "QQ 官网",
service_phone = "020-88888888",
title = "vip 会员卡",
use_custom_code = false,
source = "vqicard",
},
activate_url = "http://vqicard.com",
bonus_cleared = "年底清零",
bonus_rules = "每日签到送积分, 消费送积分",
supply_balance = false,
supply_bonus = true,
balance_rules = "不支持储值",
need_push_on_view = true,
prerogative = "vip1 特权"
};
var card = CardVoucher.Create(info, accessToken);
```

为了满足商户基于卡券本身的扩展诉求, 允许卡券内页添加 URL 跳转外链。如商户在创建卡券时添加了自定义外链, 单击此外链跳转到目标 URL 后又想记住是从哪张卡券跳转



过来的,此时就可以根据 URL 的参数获取到真实的卡券 code。注意:在特殊卡票中添加自定义 cell 字段,仅在完成数据更新后显示,即会员卡完成激活/绑定、飞机票完成在线值机、电影票完成选座信息更新后在卡券页面内显示自定义 cell。

为检测跳转外链请求来自微信,微信后台会在 URL 参数里加上如下所述的参数:

```
card_id=pR19Cs8gPMGalMS3L7hvYcGzE9fY&encrypt_code=D72wblyzmxefD1pqzF46joPgj4CtqaZuli4vXLhuiw4%3D&signature=1f039edda0135ab2a6bbf0f4a6f3f9212b141d99&openid=oR19CsyUvziSlctqDmEoMwwph4Jk
```

其中, signature 是由字段 appsecret、code、card_id 根据签名算法生成的,而 code 又是由 encrypt_code 调用解码接口获取到的。解码接口地址如下:

https://api.weixin.qq.com/card/code/decrypt?access_token=token

HTTP 请求方式: POST。

请求示例:

```
{"encrypt_code":"XXIzTtMqCxxOaawoE91+VJdsFmv7b8g0VZIZkqf4GWA60Fzpc8ksZ/5ZZ0D"}
```

请求成功后返回的数据示例如下所示:

```
{"errcode":0,"errmsg":"ok","code":"751234212312"}
```

根据上述代码创建实体类 CardCode,如代码清单 9-28 所示。

代码清单 9-28

```
namespace WxApi.ReceiveEntity
{
    public class CardCode : ErrorEntity
    {
        public string code { get; set; }
    }
}
```

因此,解码接口的实现代码如代码清单 9-29 所示。



代码清单 9-29

```
public static CardCode Decrypt(string accessToken, string encryptCode)
{
    var url =
        string.Format("https://api.weixin.qq.com/card/code/decrypt?access_
token={0}", accessToken);
    return Utils.PostResult<CardCode>(new { encrypt_code = encryptCode,
url});
}
```

签名算法是:

- (1) 将 appsecret、code、card_id 的 value 值进行字符串的字典序排序。
- (2) 将所有参数字符串拼接成一个字符串进行 SHA1 加密, 得到 signature。

简单来说就是, 将参与签名的参数的值按照字典排序拼接成字符串后进行 SHA1 加密。为了方便调用以及代码重用, 使用反射动态获取参数的值并按照算法生成签名, 如代码清单 9-30 所示。

代码清单 9-30

```
public static string GetParamSign(object obj)
{
    var type = obj.GetType();
    var pis = type.GetProperties();
    var arr = pis.Select(pi =>
    {
        var o = pi.GetValue(obj, null);
        return o == null ? "" : o.ToString();
    }).ToArray();
    //字典排序
    Array.Sort(arr);
    //拼接成字符串
    var temp = string.Join("", arr);
    return FormsAuthentication.HashPasswordForStoringInConfigFile(temp,
"SHA1");
}
```



所以，获取外链签名的实现代码如代码清单 9-31 所示。

代码清单 9-31

```
public static string GetOutLinkSign(string appsecret, string code, string cardId)
{
    return Utils.GetParamSign(new { appsecret = appsecret, code = code, card_id = cardId });
}
```

9.5 卡券投放

9.5.1 创建二维码

创建卡券后，商户可通过接口生成一张卡券二维码供用户扫码后添加卡券到卡包。接口地址如下：

https://api.weixin.qq.com/card/qrcode/create?access_token=token

HTTP 请求方式：POST。

请求的参数格式如图 9-8 所示。

```
{
  "str" action_name: "QR_CARD"
  {obj} action_info
    {obj} card
      "str" card_id: "pFS7Fjg8kV1IdDz01r4SQwMkuCKc"
      "str" code: "198374613512"
      "str" openid: "oFS7FjI0WsZ9AMZqrI80nbIq8xrA"
      "str" expire_seconds: "1800"
      0|1 is_unique_code: false
      Num outer_id: 1
}
```

图 9-8

字段详细说明如表 9-4 所示。



表 9-4

字 段	说 明	是 否 必 填
card_id	卡券 ID	是
code	指定卡券 code 码, 只能被领一次。use_custom_code 字段为 true 的卡券必须填写, 非自定义 code 不必填写	否
openid	指定领取者的 openid, 只有该用户能领取。bind_openid 字段为 true 的卡券必须填写, 非自定义 openid 不必填写	否
expire_seconds	指定二维码的有效时间, 范围是 60~1800 秒。若不填, 则默认为永久有效	否
is_unique_code	指定下发二维码, 生成的二维码随机分配一个 code, 领取不可再次扫描。填写 true 或 false。默认为 false	否
balance	红包余额, 以分为单位。红包类型必填 (LUCKY_MONEY), 其他卡券类型不填	否
outer_id	领取场景值, 用于领取渠道的数据统计, 默认值为 0, 字段类型为整型。用户领取卡券后触发的事件推送中会带上此自定义场景值	否

接口调用成功后, 返回的 JSON 字符串如下所示:

```
{"errcode":0,"errmsg":"ok","ticket":"gQG28DoAAAAAAAAAASxodHRwOi8vd2VpeGluLnFmNvbS9xL0FuWC1DNmZuVEhvMVp4NDNMNRnNRAAIEesLvUQMECAcAAA=="}
```

根据上述代码创建实体类, 如代码清单 9-32 所示。

代码清单 9-32

```
namespace WxApi.ReceiveEntity
{
    public class CardQrTicket:ErrorEntity
    {
        /// <summary>
        /// 获取二维码的 ticket。凭借此 ticket 可以在有效时间内换取二维码
        /// </summary>
        public string ticket { get; set; }
    }
}
```

最后, 在 CardVoucher 类中创建 CreateCardQrTicket 方法, 用于实现换取二维码 ticket 的方法, 如代码清单 9-33 所示。

代码清单 9-33

```
public static CardQrTicket CreateCardQrTicket(string cardId, string
```




```
accessToken, string code = null, string openid = null, int? expire = null,
bool is_unique_cide = false, int? balance = null, int outer_id = 0)
{
    var obj = new
    {
        action_name = "QR_CARD",
        action_info = new
        {
            card = new
            {
                card_id = cardId,
                code = code,
                openid = openid,
                expire_seconds = expire,
                is_unique_cide = is_unique_cide,
                outer_id = outer_id
            }
        }
    };
    var url =
        string.Format("https://api.weixin.qq.com/card/qrcode/create?access_
token={0}", accessToken);
    return Utils.PostResult<CardQrTicket>(obj, url);
}
```

获取二维码 ticket 后, 开发者可用 ticket 获取二维码图片。获取方式请参考 3.7.1 节的相关内容。

9.5.2 获取 api_ticket

api_ticket 是用于调用微信 JSAPI 的临时票据, 有效期为 7200 秒, 通过 access_token 来获取。特别注意: 由于获取 api_ticket 的 API 调用次数非常有限, 频繁刷新 api_ticket 会导致 API 调用受限, 影响自身业务, 因此开发者需在自己的服务存储与更新 api_ticket。

接口地址如下:

https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_token=ACCESS_TOKEN&type=wx_card



HTTP 请求方式: GET。

调用成功后返回的 JSON 数据如代码清单 9-34 所示。

代码清单 9-34

```
{
  "errcode":0,
  "errmsg":"ok",
  "ticket":"bxLdikRXVbTPdHSM05e5u5sUoXNKdvsdshFKA",
  "expires_in":7200
}
```

根据上述示例代码创建实体类 CardApiTicket, 如代码清单 9-35 所示。

代码清单 9-35

```
using System;
namespace WxApi.ReceiveEntity
{
    public class CardApiTicket : ErrorEntity
    {
        public string ticket { get; set; }
        /// <summary>
        /// 过期时间
        /// </summary>
        public DateTime ExpirationTime { get; set; }
        private int _expires_in;
        /// <summary>
        /// 凭证有效时间, 单位: 秒
        /// </summary>
        public int expires_in
        {
            set
            {
                ExpirationTime = DateTime.Now.AddSeconds(value / 2);
                _expires_in = value;
            }
        }
    }
}
```

最后, 在 CardVoucher 类中添加获取 api_ticket 的实现代码, 如代码清单 9-36 所示。



代码清单 9-36

```
public static CardApiTicket GetApiTicket(string accessToken)
{
    var url =
    string.Format("https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_
    token={0}&type=wx_card", accessToken);
    return Utils.GetResult<CardApiTicket>(url);
}
```

9.5.3 批量添加卡券接口

微信 JS-SDK 中提供了批量添加卡券的接口 `addCard`。此接口供商户前端网页调用，用户将一张或多张卡券添加到用户卡包。

批量添加卡券接口使用示例如图 9-9 所示。



图 9-9

调用方式如代码清单 9-37 所示。

代码清单 9-37

```
wx.addCard({
```



```
cardList: [{
    cardId: '',
    cardExt: ''
}], // 需要添加的卡券列表
success: function (res) {
    var cardList = res.cardList; // 添加的卡券列表信息
}
});
```

在上述代码中，cardExt 本身是一个 JSON 字符串，是商户为该张卡券分配的唯一性信息，包含以下字段（见表 9-5）。

表 9-5

字 段	是 否 必 填	说 明
code	否	指定的卡券 code 码，只能被领一次。use_custom_code 字段为 true 的卡券必须填写，非自定义 code 不必填写
openid	否	指定领取者的 openid，只有该用户能领取。bind_openid 字段为 true 的卡券必须填写，非自定义 openid 不必填写
timestamp	是	时间戳，商户生成从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间，且最终需要转换为字符串形式
signature	是	签名
balance	否	红包余额，以分为单位。红包类型必填（LUCKY_MONEY），其他卡券类型不填

此接口的签名方法和卡券外链的签名方法是一样的，只不过参与签名的参数不一样。此接口参与签名的参数如下：api_ticket、timestamp、card_id、code、openid、balance。根据以上字段创建实体类 CardExt，如代码清单 9-38 所示。

代码清单 9-38

```
namespace WxApi.SendEntity.Card
{
    /// <summary>
    /// 卡券扩展字段
    /// </summary>
    public class CardExt
    {
        public string code { get; set; }
        public string openid { get; set; }
        public string timestamp { get; set; }
```




```
public string signature { get; set; }  
/// <summary>  
/// 红包余额，以分为单位。红包类型必填 (LUCKY_MONEY)，其他卡券类型不填  
/// </summary>  
public string balance { get; set; }  
public string api_ticket { get; set; }  
public string card_id { get; set; }  
}  
}
```

批量添加卡券生成签名的方法如代码清单 9-39 所示。

代码清单 9-39

```
public static string GetJsApiSign(CardExt ext)  
{  
    return Utils.GetParamSign(ext);  
}
```

9.6 卡券核销

9.6.1 消耗 code

消耗 code 接口是核销卡券的唯一接口，仅支持核销有效期内的卡券；否则会返回错误码 invalid time。

接口地址如下：

https://api.weixin.qq.com/card/code/consume?access_token=TOKEN

HTTP 请求方式：POST。

请求参数如表 9-6 所示。

表 9-6

字 段	说 明	是 否 必 填
card_id	卡券 ID。创建卡券时，在 use_custom_code 填写为 true 时必填。非自定义 code 不必填写	否
code	要消耗的序列号	是



接口调用成功后,返回的示例数据如代码清单 9-40 所示。

代码清单 9-40

```
{
    "errcode":0,
    "errmsg":"ok",
    "card":{"card_id":"pFS7Fjg8kVlIdDz0lr4SQwMkuCKc"},
    "openid":"oFS7Fjl0WsZ9AMZqrI80nbIq8xrA"
}
```

根据上述示例代码创建实体类 `ConsumeRes`, 如代码清单 9-41 所示。

代码清单 9-41

```
namespace WxApi.ReceiveEntity
{
    public class ConsumeRes:ErrorEntity
    {
        public CardId card { get; set; }
        public string openid { get; set; }
    }
    public class CardId
    {
        public string card_id { get; set; }
    }
}
```

实体创建完毕后,在 `CardVoucher` 类中添加该接口的调用方法 `Consume`, 如代码清单 9-42 所示。

代码清单 9-42

```
public static ConsumeRes Consume(string accessToken, string code, string
cardId = null)
{
    var url =
        string.Format("https://api.weixin.qq.com/card/code/consume?access_
token={0}",accessToken);
    var obj = new {code = code, card_id = cardId};
```



```
return Utils.PostResult<ConsumeRes>(obj, url);
```

9.6.2 调起卡券列表并获取用户选择列表

调起适用于门店的卡券列表并获取用户选择列表的接口使用示例如图 9-10 所示。

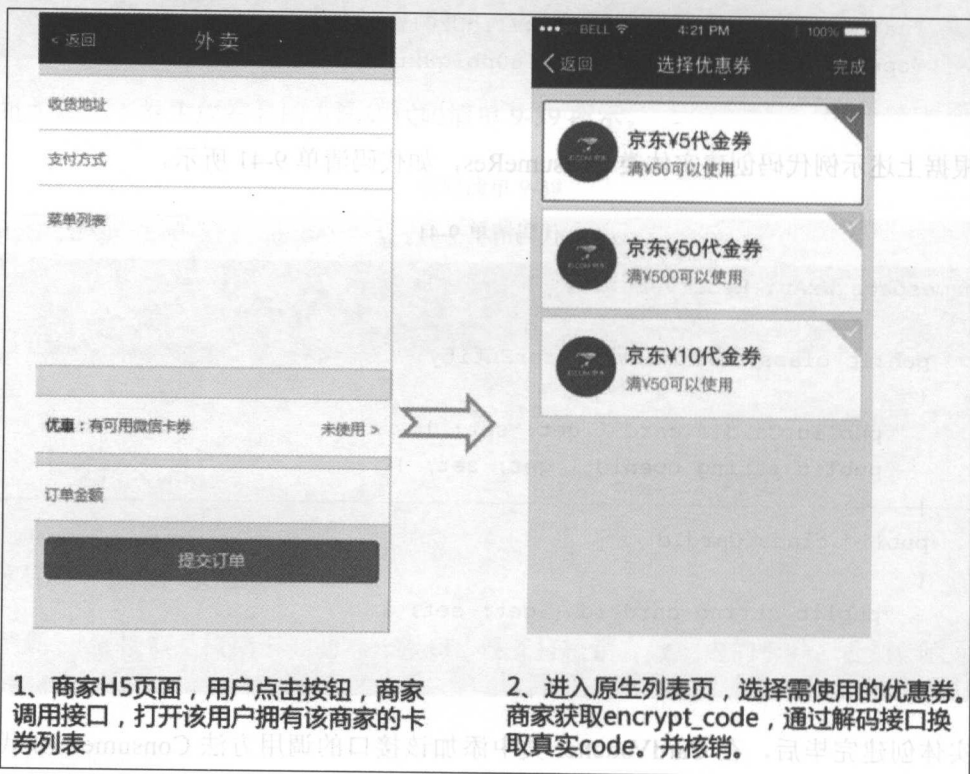


图 9-10

使用方法如代码清单 9-43 所示。

代码清单 9-43

```
wx.chooseCard({  
  shopId: '', //门店 Id。为拉取卡券列表的筛选条件之一。拉取无门店类型的卡券，不填  
  //写该字段  
  cardType: '', //卡券类型，可拉起指定类型的卡券列表。空时，默认拉起所有卡券的列表  
  cardId: '', //生成卡券时获得的 card_id，可拉起指定 id 的卡券列表。当 card_id 为
```



```
//空时默认拉起所有卡券的列表
timestamp: 0, //卡券签名时间戳
nonceStr: '', //卡券签名随机串
signType: '', //签名方式, 默认'SHA1'
cardSign: '', //卡券签名
success: function (res) {
    var cardList= res.cardList; //用户选中的卡券列表信息
}
}); //注: 不建议同时填写或不填 card_id 和 card_type
```

此接口的签名方法和卡券外链的签名方法是一样的, 只不过参与签名的参数不一样。参与签名的字段如下: api_ticket、app_id、location_id、times_tamp、nonce_str、card_id、card_type。将调用 chooseCard 方法的参数抽象成类 ChooseCard, 如代码清单 9-44 所示。

代码清单 9-44

```
namespace WxApi.SendEntity.Card
{
    public class ChooseCard
    {
        public string app_id { get; set; }
        public string location_id { get; set; }
        public string times_tamp { get; set; }
        public string nonce_str { get; set; }
        public string card_id { get; set; }
        public string card_type { get; set; }
        public string api_ticket { get; set; }
        public string signature { get; set; }
    }
}
```

所以, 生成卡券签名的方法如代码清单 9-45 所示。

代码清单 9-45

```
public static string GetChooseSign (ChooseCard param)
{
    return Utils.GetParamSign(param);
}
```




选择卡券后, JS 返回的数据如代码清单 9-46 所示。

代码清单 9-46

```
{
  "errMsg": "chooseCard:ok",
  "cardList": "[{\\"card_id\\":\\"pRl9Cs8gPMGalMS3L7hvYcGzE9fY\\",
  \\"encrypt_code\\":\\"XXIzTtMqCxxOaawoE9l+VIs/e9xkPzYvMh9s00YEUf1npwN
PWhDHlc3rqgmIEaJw\\"}]"]
}
```

请注意,在上述代码中, cardList 的类型是字符串类型。在使用的过程中,可将 cardList 的值转换成 JSON 对象。参数中的 encrypt_code 需要通过调用 9.4 节中讲到的 code 解码接口进行解码后,获取真实的 code。

最后,可以通过查看微信卡包中的卡券接口查看用户的卡券列表,调用方法如代码清单 9-47 所示。

代码清单 9-47

```
wx.openCard({
  cardList: [{
    cardId: '',
    code: ''
  }]/ 需要打开的卡券列表
});
```

9.7 卡券管理

9.7.1 删除卡券

删除卡券接口允许商户删除任意一类卡券。删除卡券后,该卡券对应已生成的领取用二维码、添加到卡包 JSAPI 均会失效。

注意:如用户在商家删除卡券前已领取一张或多张该卡券,则该卡券依旧有效。即删除卡券不能删除已被用户领取,保存在微信客户端中的卡券。



接口地址如下:

`https://api.weixin.qq.com/card/delete?access_token=TOKEN`

HTTP 请求方式: POST。

请求示例: `{"card_id": "p1Pj9jr90_SQRaVqYI239Kalerkl"}`。

因此, 接口调用方法如代码清单 9-48 所示。

代码清单 9-48

```
public static ErrorEntity Delete(string accessToken, string cardId)
{
    var obj = new { card_id = cardId };
    var url =
        string.Format("https://api.weixin.qq.com/card/delete?access_token=
{0}", accessToken);
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

9.7.2 查询 code

调用查询 code 接口可获取 code 的有效性(非自定义 code)、该 code 对应的用户 openid、卡券有效期等信息。自定义 code (use_custom_code 为 true) 的卡券调用接口时, post 数据中需包含 card_id, 非自定义 code 无须上报。

接口地址如下:

`https://api.weixin.qq.com/card/code/get?access_token=TOKEN`

HTTP 请求方式: POST。

请求示例: `{"card_id": "p1Pj9jr90_SQRaVqYI239Kalerkl", "code": "110201201245"}`。

请求成功后, 返回的数据示例如代码清单 9-49 所示。

代码清单 9-49

```
{
    "errcode": 0,
```



```
"errmsg": "ok",
"card": {
    "card_id": "pR19Cs8gPMGalMS3L7hvYcGzE9fY",
    "begin_time": 1431619200,
    "end_time": 1434211199
},
"openid": "oR19CsyUvziS1ctqDmEoMwwph4Jk"
}
```

根据上述代码创建实体类 `CodeInfo`，如代码清单 9-50 所示。

代码清单 9-50

```
using System;

namespace WxApi.ReceiveEntity
{
    /// <summary>
    /// code 信息
    /// </summary>
    public class CodeInfo
    {
        public CardValid card { get; set; }
        public string openid { get; set; }
    }
    public class CardValid
    {
        public string card_id { get; set; }
        public int begin_time { get; set; }

        public DateTime BeginTime
        {
            get { return Utils.UnixTimeToTime(begin_time.ToString()); }
        }
        public DateTime EndTime
        {
            get { return Utils.UnixTimeToTime(end_time.ToString()); }
        }
    }
}
```



```
public int end_time { get; set; }  
}  
}
```

实体创建完毕后,在 CardVoucher 类中添加实现此接口的方法,如代码清单 9-51 所示。

代码清单 9-51

```
public static CodeInfo QueryCode(string accessToken, string code, string  
cardId="")  
{  
    var obj = new {code=code, card_id = cardId };  
    var url =  
        string.Format("https://api.weixin.qq.com/card/code/get?access_token=  
{0}", accessToken);  
    return Utils.PostResult<CodeInfo>(obj, url);  
}
```

注意:固定时长有效期会根据用户实际领取时间转换,如用户 2013 年 10 月 1 日领取,固定时长有效期为 90 天,即有效时间为 2013 年 10 月 1 日至 12 月 29 日。

9.7.3 批量查询卡列表

此接口的功能是查询商户下所有的卡券列表。接口地址如下:

https://api.weixin.qq.com/card/batchget?access_token=TOKEN

HTTP 请求方式: POST。

请求示例: {"offset": 0, "count": 10}。

其中, offset 表示的是查询卡列表的起始偏移量,从 0 开始,即 offset: 5 是指从列表里的第 6 个开始读取。count 表示的是需要查询的卡片数量(数量最大为 50)。

请求成功后,返回的数据示例如代码清单 9-52 所示。

代码清单 9-52

```
{  
    "errcode": 0,  
    "errmsg": "ok",
```




```
"card_id_list":["ph_gmt7cUVrlRk8swPwx7aDyF-pg"],  
"total_num":1  
}
```

在上述代码中, `total_num` 表示的是该商户名下的 `card_id` 总数。根据上述示例代码创建实体类 `CardIds`, 如代码清单 9-53 所示。

代码清单 9-53

```
using System.Collections.Generic;  
namespace WxApi.ReceiveEntity  
{  
    public class CardIds:ErrorEntity  
    {  
        public List<string> card_id_list { get; set; }  
        public int total_num { get; set; }  
    }  
}
```

实体创建完毕后, 在 `CardVoucher` 类中添加实现此接口的方法 `GetCardIdList`, 如代码清单 9-54 所示。

代码清单 9-54

```
public static CardIds GetCardIdList(int offset, int count, string  
accessToken)  
{  
    var url =  
        string.Format("https://api.weixin.qq.com/card/batchget?access_token=  
{0}", accessToken);  
    var obj = new { offset = offset, count = count };  
    return Utils.PostResult<CardIds>(obj, url);  
}
```

9.7.4 查询卡券详情

此接口是为修改卡券做数据支持的, 在调用修改卡券接口之前需要通过此接口获取卡券的详细信息。接口地址如下:



`https://api.weixin.qq.com/card/get?access_token=TOKEN`

HTTP 请求方式: POST。

请求示例: `{"card_id": "pFS7Fjg8kV1IdDz01r4SQwMkuCKc"}`。

请求成功后的响应数据的 JSON 格式如图 9-11 所示。

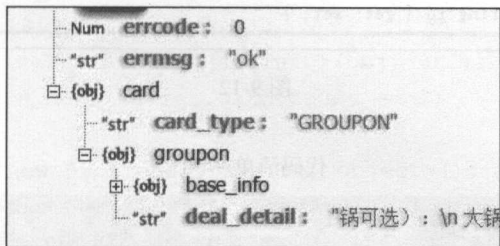


图 9-11

图 9-11 中与请求 `card_id` 对应的卡券类型是团购券, 所以 `card_type` 的类型为 `GROUPON`。从图 9-11 中可以看出, `card` 对象包含两个属性 `card_type` 和与 `card_type` 相关联的 `groupon` 对象, 但这个相关联的对象名是随着 `card_type` 变化的, 这种表示形式就无法使用常规的映射实体类来处理了。在这里, 我们可以使用和添加卡券一样的处理方式进行处理, 即使用 `Dictionary` 方式处理。首先创建实体类 `CardInfo`, 如代码清单 9-55 所示。

代码清单 9-55

```

using System.Collections.Generic;
namespace WxApi.ReceiveEntity
{
    public class CardInfo:ErrorEntity
    {
        public Dictionary<string, object> card { get; set; }
    }
}

```

另外, `base_info` 对象的属性列表和创建卡券时的属性列表基本保持一致, 但比创建卡券时多了一些属性: 一个 `status`, 表示的是卡券的状态; 另一个是 `id`, 表示的是卡券的 `card_id`。所以需要在 `BaseInfo` 类中添加这两个属性, 如图 9-12 所示。

卡券的状态是个枚举类型, 如代码清单 9-56 所示。



```

1 个引用
public Sku sku { get; set; }
/// <summary>
/// 获取卡券详情时的字段，表示卡券的状态。 创建卡券时无效
/// </summary>
0 个引用
public CardStatus? status { get; set; }
/// <summary>
/// 获取卡券详情时的字段，表示卡券的card_id。 创建卡券时无效
/// </summary>
0 个引用
public string id { get; set; }
}

```

图 9-12

代码清单 9-56

```

public enum CardStatus
{
    /// <summary>
    /// 待审核
    /// </summary>
    CARD_STATUS_NOT_VERIFY,
    /// <summary>
    /// 审核失败
    /// </summary>
    CARD_STATUS_VERIFY_FALL,
    /// <summary>
    /// 通过审核
    /// </summary>
    CARD_STATUS_VERIFY_OK,
    /// <summary>
    /// 卡券被用户删除
    /// </summary>
    CARD_STATUS_USER_DELETE,
    /// <summary>
    /// 在公众平台投放过的卡券
    /// </summary>
    CARD_STATUS_DISPATCH
}

```

在此接口调用的时候，首先将返回值转换成 Dictionary<string, object>。由于转换成 object 时 JSON 的数据是不做任何处理的，在调用的时候无法转换成真实的类型，因此需要



根据返回的 `card_type` 的类型转换成真实的类型后再返回，如代码清单 9-57 所示。

代码清单 9-57

```
public static CardInfo GetCardInfo(string accessToken, string cardId)
{
    var obj = new { card_id = cardId };
    var url = string.Format("https://api.weixin.qq.com/card/get?access_token={0}", accessToken);
    var ret = Utils.PostResult<CardInfo>(obj, url);
    if (ret.card != null && ret.card.Count == 2)
    {
        var dic = new Dictionary<string, object>();
        var card_type = ret.card["card_type"].ToString();
        dic.Add("card_type", card_type);
        var info = ret.card[card_type.ToLower()].ToString();
        switch (ret.card["card_type"].ToString())
        {
            case "GENERAL_COUPON":
                dic.Add(card_type, JsonConvert.DeserializeObject<General_Coupon>(info)); break;
            case "GROUPON":
                dic.Add(card_type, JsonConvert.DeserializeObject<Groupon>(info)); break;
            case "DISCOUNT":
                dic.Add(card_type, JsonConvert.DeserializeObject<Discount>(info)); break;
            case "GIFT":
                dic.Add(card_type, JsonConvert.DeserializeObject<Gift>(info)); break;
            case "CASH":
                dic.Add(card_type, JsonConvert.DeserializeObject<Cash>(info)); break;
            case "MEMBER_CARD":
                dic.Add(card_type, JsonConvert.DeserializeObject<Member_Card>(info)); break;
            case "SCENIC_TICKET":
                dic.Add(card_type, JsonConvert.DeserializeObject<Scenic_Ticket>(info)); break;
        }
    }
}
```




```
        case "MOVIE_TICKET":
            dic.Add(card_type, JsonConvert.DeserializeObject<Movie_Ticket>
                (info)); break;
        case "BOARDING_PASS":
            dic.Add(card_type, JsonConvert.DeserializeObject
                <Boarding_Pass>(info)); break;
        case "MEETING_TICKET":
            dic.Add(card_type, JsonConvert.DeserializeObject
                <Meeting_Ticket>(info)); break;
    }
    ret.card = dic;
}
return ret;
}
```

9.7.5 事件推送

卡券通过审核、卡券被用户领取、卡券被用户删除均会触发事件推送，该事件将发送至开发者填写的回调 URL 中。处理方式与处理用户发送消息、自定义菜单等消息是一样的，具体的处理方式请参考 3.3 节，限于篇幅，本节就不再赘述了。下面分别介绍各个与卡券相关的事件推送。

1. 审核事件推送

生成的卡券通过审核时，微信会把这个事件推送到开发者填写的 URL。推送 XML 数据包示例如代码清单 9-58 所示。

代码清单 9-58

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[FromUser]]></FromUserName>
  <CreateTime>123456789</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[card_pass_check]]></Event> //不通过为 card_not_pass_check
  <CardId><![CDATA[cardid]]></CardId>
</xml>
```

在接收到上述 XML 数据后，可以根据 CardId 和事件类型去修改商户端数据库中保存



的卡券信息状态。也可以根据具体的逻辑实现更好的功能，如卡券审核通过后推送消息告诉相关的负责人。

2. 领取卡券事件推送

用户在领取卡券时，微信会把这个事件推送到开发者填写的 URL。推送 XML 数据包示例如代码清单 9-59 所示。

代码清单 9-59

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[FromUser]]></FromUserName>
  <FriendUserName><![CDATA[FriendUser]]></FriendUserName>
  <CreateTime>123456789</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[user_get_card]]></Event>
  <CardId><![CDATA[cardid]]></CardId>
  <IsGiveByFriend>1</IsGiveByFriend>
  <UserCardCode><![CDATA[12312312]]></UserCardCode>
  <OuterId>0</OuterId>
</xml>
```

参数说明如表 9-7 所示。

表 9-7

参 数	说 明
CardId	卡券 ID
IsGiveByFriend	是否为转赠，1 代表“是”，0 代表“否”
UserCardCode	code 序列号。自定义 code 及非自定义 code 的卡券被领取后都支持事件推送
OuterId	领取场景值，用于领取渠道数据统计。可在生成二维码接口及添加 JSAPI 接口中自定义该字段的整型值

3. 删除卡券事件推送

用户在删除卡券时，微信会把这个事件推送到开发者填写的 URL。推送 XML 数据包示例如代码清单 9-60 所示。

代码清单 9-60

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
```



```
<FromUserName><![CDATA[FromUser]]></FromUserName>
<CreateTime>123456789</CreateTime>
<MsgType><![CDATA[event]]></MsgType>
<Event><![CDATA[user_del_card]]></Event>
<CardId><![CDATA[cardid]]></CardId>
<UserCardCode><![CDATA[12312312]]></UserCardCode>
</xml>
```

参数说明如表 9-8 所示。

表 9-8

参 数	说 明
CardId	卡券 ID
UserCardCode	商户自定义 code 值。非自定义 code 推送为空串

4. 进入会员卡事件推送

用户在进入会员卡时，微信会把这个事件推送到开发者填写的 URL。推送 XML 数据包示例如代码清单 9-61 所示。

代码清单 9-61

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[FromUser]]></FromUserName>
  <CreateTime>123456789</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[user_view_card]]></Event>
  <CardId><![CDATA[cardid]]></CardId>
  <UserCardCode><![CDATA[12312312]]></UserCardCode>
</xml>
```

5. 核销事件推送

卡券被核销时，微信会把这个事件推送到开发者填写的 URL。推送 XML 数据包示例如代码清单 9-62 所示。

代码清单 9-62

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
```



```
<FromUserName><![CDATA[FromUser]]></FromUserName>
<CreateTime>123456789</CreateTime>
<MsgType><![CDATA[event]]></MsgType>
<Event><![CDATA[user_consume_card]]></Event>
<CardId><![CDATA[cardid]]></CardId>
<UserCardCode><![CDATA[12312312]]></UserCardCode>
</xml>
```

9.7.6 更改 code

为确保转赠后的安全性，微信允许自定义 code 的商户对已下发的 code 进行更改。

注意：为避免用户疑惑，建议仅在发生转赠行为后（发生转赠后，微信会通过事件推送的方式告知商户被转赠的卡券 code）对用户的 code 进行更改。

接口地址如下：

https://api.weixin.qq.com/card/code/update?access_token=Token

HTTP 请求方式：POST。

POST 参数说明如表 9-9 所示。

表 9-9

参 数	说 明
code	卡券的 code 编码
card_id	卡券 ID
new_code	新的卡券 code 编码

根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-63 所示。

代码清单 9-63

```
public static ErrorEntity UpdateCode(string accessToken, string cardId,
string code, string new_code)
{
    var url =
string.Format("https://api.weixin.qq.com/card/code/update?access_token=
{0}", accessToken);
```




```
var obj = new { code = code, card_id = cardId, new_code = new_code };  
return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

9.7.7 设置卡券失效接口

为满足改票、退款等异常情况，可调用卡券失效接口将用户的卡券设置为失效状态。

注意：设置卡券失效的操作不可逆，即无法将设置为失效的卡券调回有效状态，商家须慎重调用该接口。

接口地址如下：

https://api.weixin.qq.com/card/code/unavailable?access_token=TOKEN

HTTP 请求方式：POST。

POST 参数说明如表 9-10 所示。

表 9-10

参 数	说 明
code	卡券的 code 编码
card_id	自定义 code 的卡券必填。非自定义 code 的卡券不填

根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-64 所示。

代码清单 9-64

```
public static ErrorEntity Unavailable(string accessToken, string code,  
string cardId="")  
{  
    var url =  
string.Format("https://api.weixin.qq.com/card/code/unavailable?  
access_token={0}", accessToken);  
    var obj = new { code = code, card_id = cardId };  
    return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

9.7.8 更改卡券信息接口

支持更新所有卡券类型的部分通用字段及特殊卡券（会员卡、飞机票、电影票、会议



门票) 中特定字段的信息。

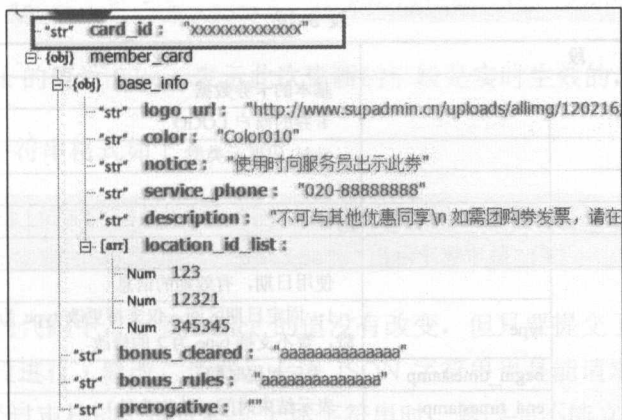
注意：更改卡券的部分字段后会重新提交审核，更新成功后可通过调用查看卡券详情接口核查更新结果。在调用接口时，对不允许修改的卡券字段进行修改是无效的。许多开发者在调用该接口时会填入 `brandname` 等不支持修改的字段，导致更新不成功。

接口地址如下：

`https://api.weixin.qq.com/card/update?access_token=TOKEN`

HTTP 请求方式：POST。

请求的数据格式如图 9-13 所示。



如图 9-13

从图 9-13 中可以看出，更改卡券信息接口 POST 的数据格式和新增卡券接口的数据格式基本一致。可更新的参数列表如表 9-11 所示。

表 9-11

字 段			说 明	是 否 过 审
card_id	member_card		卡券 ID	否
		bonus_cleared	积分清零规则	是
		bonus_rules	积分规则	是
		balance_rules	储值说明	否
		prerogative	特权说明	否



续表

字 段			说 明	是 否 过 审
card_id	scenic_ticket	guide_url	导览图 url	否
	movie_ticket	detail	电影票详情	是
	boarding_pass	departure_time	起飞时间。UNIX 时间戳格式	否
		landing_time	降落时间。UNIX 时间戳格式	否
		gate	登机口。如发生登机口变更, 则建议商家实时调用该接口变更	否
		boarding_time	登机时间, 只显示“时分”, 不显示日期, 按时间戳格式填写。如发生登机时间变更, 则建议商家实时调用该接口变更	否
	meeting_ticket	map_url	会场导览图	否

base_info 字段描述如表 9-12 所示。

表 9-12

字 段			说 明	是 否 过 审
base_info			基本的卡券数据	
	logo_url		卡券的商户 LOGO	是
	code_type		code 码展示类型	否
	color			否
	notice			是
	description			是
	date_info		使用日期, 有效期的信息	
		type	1: 固定日期区间, 仅支持更改 type 为 1 的时间戳, 暂不支持 type 为 2 的修改	是
		begin_timestamp	表示起用时间	否
		end_timestamp	表示结束时间 (单位为秒)	否
	location_id_list		门店位置 ID。商户需在 mp 平台上录入门店信息或调用批量导入门店信息接口获取门店位置 ID	否
	can_share		是否可分享	否
	can_give_friend		是否可转赠	否
	get_limit		每人最大领取次数。若不填写, 则默认等于 quantity	否
	service_phone		客服电话	否
	custom_url_name		商户自定义入口名称, 与 custom_url 字段共同使用, 长度限制在 5 个汉字内	否
	custom_url		商户自定义入口跳转外链的地址链接, 跳转页面内容需与自定义 cell 名称保持匹配	否
	custom_url_sub_title		显示在入口右侧的 tips, 长度限制在 6 个汉字内	否
	promotion_url_name		营销场景的自定义入口	否
	promotion_url		入口跳转外链的地址链接	否
	promotion_url_sub_title		显示在入口右侧的 tips, 长度限制在 6 个汉字内	否



注意：在表 9-12 中，是否过审指的是字段更新时是实时生效还是需要通过审核。除上述所有字段外，对其他字段的更新都是无效的。更新时只传入需要更新的字段，不更新的字段请勿传入。这样就能避免更新不需要审核的字段，这种更新可以实时生效。比如说某个折扣券的 notice 值为“出示卡券审核”，service_phone 的值为“027-00000000”，当需要更新 service_phone 的值为“027-00010001”时，正确的 POST 的 JSON 字符串应如下所示：

```
{"card_id":"pR19Cs44qa6gDYHmKdH9noM7p5o","discount":{"base_info":
{"service_phone":"027-00010001"}}}
```

请求成功后返回的数据如下：

```
{"errcode":0,"errmsg":"ok","send_check":false}
```

其中 send_check 的值为 false，表示此次更新的字段是实时生效的，无须审核。

错误的 JSON 字符串格式如下：

```
{"card_id":"pR19Cs44qa6gDYHmKdH9noM7p5o","discount":{"base_info":
{"service_phone":"027-00010001","notice":"出示卡券审核"}}}
```

在上述错误示例代码中，尽管 notice 的值没有改变，但只要提交了该字段，微信服务器都会认为对此字段进行了修改。当然，上述 JSON 字符串也是能请求成功的，但是由于 notice 字段是需要经过审核的，因此提交上述字符串时，修改并不能立即生效。

现在问题来了，如果使用创建卡券的那种实体赋值后再转换成 JSON 字符串的方式，转换 JSON 的过程中并不能移除不需要的字段。尽管可以只给需要更新的字段赋值，但未赋值的字段也会初始化字段类型默认的值，如 int 类型的参数会变成 0，string 类型的会变成空字符。

根据上述可更新参数列表封装成枚举，在调用的时候就可以很方便地找到需要更新的字段。其结构如图 9-14 所示。

限于篇幅，在此就不贴出详细代码了，读者可参考随书源码。根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-65 所示。



```

namespace WxApi.SendEntity.Card
{
    /// <summary>
    /// 卡券可更新基础字段枚举
    /// </summary>
    1 个引用
    public enum BaseInfoKey...

    /// <summary>
    /// 卡券日期字段枚举。使用日期，有效期时间修改仅
    /// </summary>
    0 个引用
    public enum DateInfoKey...

    /// <summary>
    /// 会员卡可修改字段枚举
    /// </summary>
    0 个引用
    public enum MemberCardKey...
}
    
```

图 9-14

代码清单 9-65

```

/// <summary>
/// 更新卡券字段（只包括基础信息字段）
/// </summary>
/// <param name="accessToken"></param>
/// <param name="cardId"></param>
/// <param name="cardType">卡券类型</param>
/// <param name="bik">字段键值对</param>
/// <returns></returns>
public static ErrorEntity UpdateCardField(string accessToken, string
cardId, CardType cardType, Dictionary<BaseInfoKey, object> bik)
{
    var dic = new Dictionary<string, object>();
    dic.Add("card_id", cardId);
    dic.Add(cardType.ToString().ToLower(), new { base_info = bik });
    return UpdateCardField(accessToken, dic);
}
/// <summary>
/// 更新卡券字段（特殊卡券更新。包括共有的基础字段和私有字段）
/// </summary>
    
```



```
/// <param name="accessToken"></param>
/// <param name="cardId"></param>
/// <param name="cardType">卡券类型</param>
/// <param name="pik">字段键值对</param>
/// <returns></returns>
public static ErrorEntity UpdateCardField(string accessToken, string
cardId, CardType cardType, Dictionary<Enum, object> pik)
{
    var dic = new Dictionary<string, object>();
    dic.Add("card_id", cardId);
    dic.Add(cardType.ToString().ToLower(), pik);
    return UpdateCardField(accessToken, dic);
}
private static ErrorEntity UpdateCardField(string accessToken, object obj)
{
    var url = string.Format("https://api.weixin.qq.com/card/update?access_
token={0}", accessToken);
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

图 9-15 所示为上述方法的调用示例。

```
var accessToken = AccessTokenBox.GetTokenValue("wx891ee9903ba8d74e", "dd93fb224cc735086a6c24fea55a7dbc");
var bi = new Dictionary<BaseInfoKey, object>();
var pi = new Dictionary<Enum, object>();
bi.Add(BaseInfoKey.color, "Color010");
bi.Add(BaseInfoKey.custom_url_name, "测试");
CardVoucher.UpdateCardField(accessToken, "pR19Cs65m56sbQDvg0PfbDmh0RBc", CardType.DISCOUNT, pi);
```

图 9-15

9.7.9 库存修改接口

修改库存就是对 sku 的值进行修改。接口地址如下：

https://api.weixin.qq.com/card/modifystock?access_token=TOKEN

HTTP 请求方式：POST。

POST 参数说明如表 9-13 所示。



表 9-13

参 数	说 明
card_id	
increase_stock_value	增加多少库存, 可以不填或填 0
reduce_stock_value	减少多少库存, 可以不填或填 0

根据上述接口说明在 CardVoucher 中添加调用此接口的方法, 如代码清单 9-66 所示。

代码清单 9-66

```
public static ErrorEntity ModifyStock(string accessToken, string cardId, int sum)
{
    var url =
        string.Format("https://api.weixin.qq.com/card/modifystock?access_token={0}", accessToken);
    var dic = new Dictionary<string, string>();
    var obj = new
    {
        increase_stock_value = sum > 0 ? sum : 0,
        reduce_stock_value = sum > 0 ? 0 : Math.Abs(sum),
        card_id = cardId
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

在上述代码的参数中, sum 表示的是修改的库存数量。正数为增加, 负数为减少。

9.8 特殊卡票操作

9.8.1 会员卡

商户调用接口创建会员卡获取 card_id, 并将会员卡下发给用户, 用户领取后需激活/绑定, 更新会员卡编号及积分信息。会员卡暂不支持转赠。激活/绑定会员卡支持以下两种方式:



(1) 用户单击卡券内的“bind_old_card_url”、“activate_url”跳转到商户自定义的 H5 页面, 填写相关身份认证信息后, 商户调用接口, 完成激活/绑定。

(2) 商户已知用户身份或无须进行绑定等操作, 用户领取会员卡后, 商户后台即时调用“激活绑定会员卡”接口, 更新会员卡编号及积分信息。

注意: code 与会员卡编号 membership_number 为一一对应关系。商户在调用相关查询、交易接口时, 需使用 code 字段。

激活/绑定会员卡接口地址如下:

https://api.weixin.qq.com/card/memberscard/activate?access_token=TOKEN

HTTP 请求方式: POST。

请求的参数列表如表 9-14 所示。

表 9-14

字 段	说 明	是 否 必 填
init_bonus	初始积分, 不填为 0	否
init_balance	初始余额, 不填为 0	否
membership_number	必填, 会员卡编号, 作为序列号显示在用户的卡包里	是
code	创建会员卡时获取的 code	是
card_id	卡券 ID。自定义 code 的会员卡必填 card_id, 非自定义 code 的会员卡不必填	否

根据上述接口说明在 CardVoucher 中添加调用此接口的方法, 如代码清单 9-67 所示。

代码清单 9-67

```
public static ErrorEntity ActivateMemberCard(string accessToken,
string membership_number, string code, int init_bonus = 0, int init_balance
= 0, string cardId="")
{
    var url =
        string.Format("https://api.weixin.qq.com/card/memberscard/activate?
access_token={0}", accessToken);
    var obj = new
    {
        init_bonus = init_bonus,
```




```
init_balance = init_balance,  
membership_number = membership_number,  
code = code,  
card_id = cardId  
};  
return Utils.PostResult<ErrorEntity>(obj, url);  
}
```

激活会员卡后，在会员卡每次交易后，积分与余额变更需通过接口通知微信，便于后续消息通知及其他扩展功能。接口地址如下：

https://api.weixin.qq.com/card/memberscard/updateuser?access_token=TOKEN

HTTP 请求方式：POST。

请求的参数列表如表 9-15 所示。

表 9-15

字 段	说 明	是 否 必 填
add_bonus	需要变更的积分，扣除积分用“-”表示	否
record_bonus	商家自定义积分消耗记录，不超过 14 个汉字	否
add_balance	需要变更的余额，扣除金额用“-”表示。单位为分	否
membership_number	必填，会员卡编号，作为序列号显示在用户的卡包里	否
record_balance	商家自定义金额消耗记录，不超过 14 个汉字	否
code	创建会员卡时获取的 code	是
card_id	卡券 ID。自定义 code 的会员卡必填 card_id，非自定义 code 的会员卡不必填	否

接口请求成功后，返回的数据如代码清单 9-68 所示。

代码清单 9-68

```
{  
  "errcode":0,  
  "errmsg":"ok",  
  "result_bonus": 100,  
  "result_balance": 200  
  "openid":"oFS7Fj10WsZ9AMZqrI80nbIq8xrA"  
}
```



根据上述返回数据创建实体类 `MemberCardResult`，如代码清单 9-69 所示。

代码清单 9-69

```
namespace WxApi.ReceiveEntity
{
    public class MemberCardResult : ErrorEntity
    {
        /// <summary>
        /// 当前用户积分总额
        /// </summary>
        public int result_bonus { get; set; }
        /// <summary>
        /// 当前用户预存总金额
        /// </summary>
        public int result_balance { get; set; }
        /// <summary>
        /// 用户 openid
        /// </summary>
        public int openid { get; set; }
    }
}
```

实体创建完后，根据上述接口说明在 `CardVoucher` 中添加调用此接口的方法，如代码清单 9-70 所示。

代码清单 9-70

```
public static MemberCardResult UpdateMemberCard(string accessToken, string
code, int add_bonus = 0, string record_bonus = "", int add_balance = 0, string
record_balance = "", string card_id = "")
{
    var url =
        string.Format("https://api.weixin.qq.com/card/membercard/updateuser?
access_token={0}", accessToken);
    var obj = new
    {
        code = code,
        add_balance = add_balance,
        record_balance = record_balance,
```



```
        add_bonus = add_bonus,
        record_bonus = record_bonus,
        card_id = card_id
    };
    return Utils.PostResult<MemberCardResult>(obj, url);
}
```

9.8.2 电影票

电影票券主要分为以下两种：

(1) 电影票兑换券，归属于团购券。

(2) 选座电影票，在购买时需要选定电影、场次、座位，具备较强的时效性和特殊性，此类电影票券即本节中的电影票。

使用场景：用户单击商户 H5 页面提供的 JSAPI（添加到卡包 JSAPI）后，商户根据用户电影票信息，调用接口创建卡券，获取 card_id 后，将带有唯一 code 的电影票下发给用户，用户领取后通过接口（更新电影票）更新用户选座信息。

更新电影票的接口地址如下：

https://api.weixin.qq.com/card/movieticket/updateuser?access_token=TOKEN

HTTP 请求方式：POST。

请求的参数列表如表 9-16 所示。

表 9-16

字 段	说 明	是 否 必 填
ticket_class	电影票的类别，如 2D、3D	是
show_time	电影放映时间对应的时间戳	是
duration	放映时长，填写整数	是
screening_room	该场电影的影厅信息	是
seat_number	座位号。类型为数组。如["5 排 14 号", "5 排 15 号"]	是
code	电影票的序列号	是
card_id	电影票 card_id。自定义 code 的电影票为必填，非自定义 code 的电影票不必填	否

根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-71 所示。



代码清单 9-71

```
public static ErrorEntity UpdateMovieTicket(string accessToken, string code,
string ticket_class, int show_time, int duration, string screening_room,
string[] seat_number, string card_id="")
{
    var url =
        string.Format("https://api.weixin.qq.com/card/movieticket/
updateuser?access_token={0}", accessToken);
    var obj = new
    {
        code = code,
        card_id = card_id,
        ticket_class = ticket_class,
        show_time = show_time,
        duration = duration,
        screening_room = screening_room,
        seat_number = seat_number
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

9.8.3 飞机票在线值机

接口地址如下:

https://api.weixin.qq.com/card/boardingpass/checkin?access_token=TOKEN

HTTP 请求方式: POST。

请求的参数列表如表 9-17 所示。

表 9-17

字 段	说 明	是 否 必 填
passenger_name	乘客姓名, 上限为 15 个汉字	是
class	舱等, 如头等舱等, 上限为 5 个汉字	是
seat	乘客座位号	是
etkt_bnr	电子客票号, 上限为 14 个数字	是
qrcode_data	二维码数据。乘客用于值机的二维码字符串, 微信会通过此数据为用户生成值机用的二维码	是



续表

字 段	说 明	是 否 必 填
is_cancel	是否取消值机。填写 true 或 false。true 代表取消，如填写 true，则上述字段（如 calss 等）均不做判断，机票返回未值机状态，乘客可重新值机。默认填写 false	否
code	机票的序列号	是
card_id	需办理值机的机票 card_id。自定义 code 的飞机票为必填	否

根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-72 所示。

代码清单 9-72

```
public static ErrorEntity CheckIn(string accessToken, string code, string
passenger_name,string @class, string etkt_bnr, string seat = "", string
qrcode_data = "", bool is_cancel = false, string card_id = "")
{
    var url =
        string.Format("https://api.weixin.qq.com/card/boardingpass/checkin?
access_token={0}", accessToken);
    var obj = new
    {
        code = code,
        card_id = card_id,
        passenger_name = passenger_name,
        @class = @class,
        seat = seat,
        etkt_bnr = etkt_bnr,
        qrcode_data = qrcode_data,
        is_cancel = is_cancel
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```

在上述代码中，由于舱等的字段名为 class，class 又是 C#的保留字，因此使用@字符转义一下。

9.8.4 更新会议门票

此接口支持更新入场时间、区域、座位等信息。接口地址如下：



https://api.weixin.qq.com/card/meetingticket/updateuser?access_token=TOKEN

HTTP 请求方式: POST。

请求的参数列表如表 9-18 所示。

表 9-18

字 段	说 明	是 否 必 填
zone	区域	是
entrance	入口	是
seat_number	座位号	是
begin_time	开场时间。UNIX 时间戳	是
end_time	结束时间。UNIX 时间戳	是
code	用户的门票唯一序列号	是
card_id	要更新门票序列号所述的 card_id, 生成券时 use_custom_code 填写 true 时必填	否

根据上述接口说明在 CardVoucher 中添加调用此接口的方法, 如代码清单 9-73 所示。

代码清单 9-73

```
public static ErrorEntity UpdateMeeting(string accessToken, string code, int
begin_time, int end_time, string zone = "", string entrance = "", string
seat_number = "", string card_id = "")
{
    var url =
        string.Format("https://api.weixin.qq.com/card/meetingticket/
updateuser?access_token={0}", accessToken);
    var obj = new
    {
        code = code,
        card_id = card_id,
        zone = zone,
        entrance = entrance,
        seat_number = seat_number,
        begin_time = begin_time,
        end_time = end_time
    };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```



9.9 设置测试白名单

由于卡券有审核要求,未通过审核的卡券是无法领取的,如图 9-16 所示。为方便公众账号调试,可以设置一些测试账号,这些账号可领取未通过审核的卡券,体验整个流程。

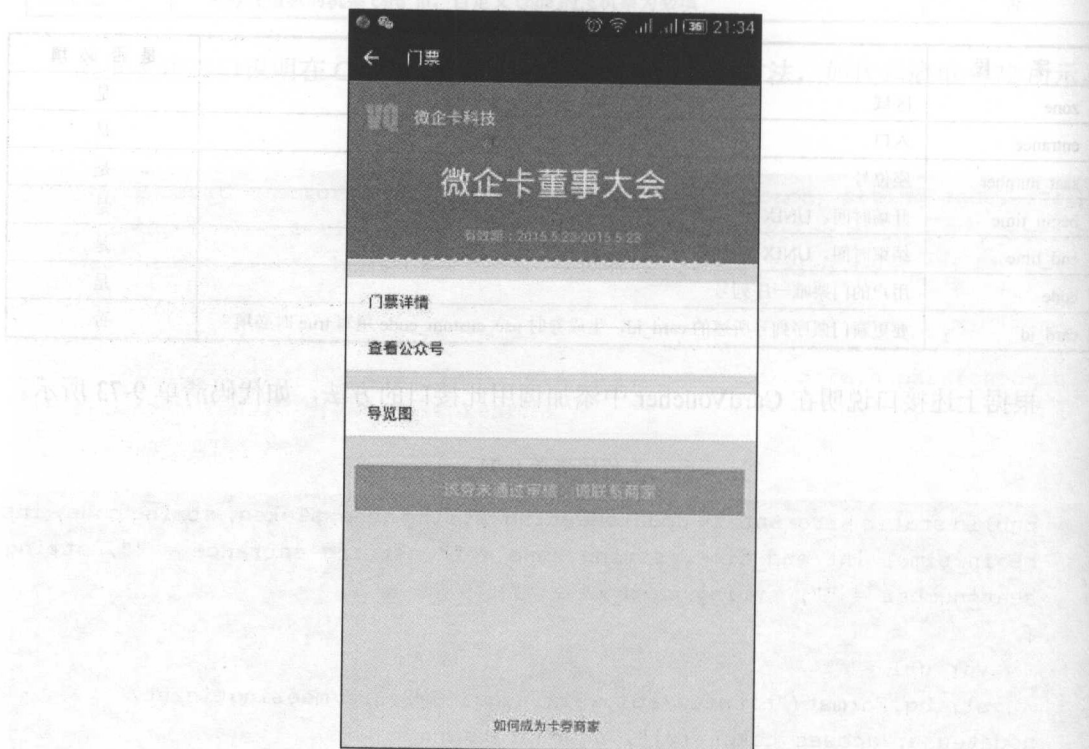


图 9-16

注意:同时支持“openid”、“username”两种字段设置白名单,总数上限为 10 个。

接口地址如下:

https://api.weixin.qq.com/card/testwhitelist/set?access_token=TOKEN

HTTP 请求方式: POST。

请求的参数列表如表 9-19 所示。



表 9-19

字 段	说 明	是 否 必 填
openid	测试的 openid 列表	否
username	测试的微信号列表	否
openid 和 username 不能同时为空		

根据上述接口说明在 CardVoucher 中添加调用此接口的方法，如代码清单 9-74 所示。

代码清单 9-74

```
public static ErrorEntity TestWhite(string accessToken, List<string>
openidList = null, List<string> userList = null)
{
    var url =
        string.Format("https://api.weixin.qq.com/card/testwhitelist/set?
access_token={0}", accessToken);
    var obj = new { openid = openidList, username = userList };
    return Utils.PostResult<ErrorEntity>(obj, url);
}
```


第 10 章 应用案例

10.1 微信扫一扫登录 PC 网站

微信登录 PC 网站官方推荐的做法是使用 OAuth2.0 授权，但前提是需要先在开放平台注册开发者账号，然后将网站绑定在开发者账号下。在用户登录时，微信扫描网页授权 URL 生成的 URL 后，跳转到授权页面，询问用户是否登录，用户确认登录后，则登录成功。这种方式的弊端是用户无须关注公众号即可实现登录或注册操作，但大部分运营者却希望更多用户关注自己的公众号。所以在本案例中并不使用此种方式进行登录，而是使用带参数二维码的机制变相实现登录操作。

在 3.7 节中已经讲述了带参数二维码的机制，其中临时二维码非常适合实现本案例的登录操作。实现原理是：用户选择微信登录后，生成一个临时二维码（二维码参数需唯一，推荐使用时间戳），将此二维码的参数信息保存在数据库或全局的变量中，同时监听二维码被扫事件，二维码被扫之后可以获取到扫描者的 openid，最后通过 openid 即可获取到用户的详细信息，即实现了登录的流程。

详细实现过程如下：

(1) 新建页面 `qrLogin.aspx`，添加 `jquery` 引用到页面中。然后在页面中添加一个按钮和一个 `img` 标签，如图 10-1 所示。

(2) 添加按钮的 `click` 事件代码。单击按钮后，异步请求获取带参数二维码路径和参数值，并将 `img` 标签的 `src` 属性的值设置为获取到的二维码路径，启动监听扫描事件。前台



JS 代码如代码清单 10-1 所示。

```
<head runat="server">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <script src="http://apps.bdimg.com/libs/jquery/1.8.3/jquery.min.js"></script>
</head>
<body>
  <input type="button" id="login" value="微信登录" />
  <img />
</body>
```

图 10-1

代码清单 10-1

```
<script>
  var time;
  var key;
  $(function () {
    //生成带参数二维码
    $("#login").click(function () {
      $.get(location.href + "?action=loginqr", function (data) {
        $("img").attr("src", data.qrurl).css({ "height": 300,
          "width": 300 });
        key = data.key;
        setTimeout(function () {
          time = setInterval(res, 1000);
        }, 3000);

        }, "json");
      });
    });
    //监听二维码扫描事件
    var res = function () {
      $.get(location.href + "?action=checkscan&key=" + key, function
        (data) {
          if (data.status == 1) {
            clearInterval(time);
            alert(JSON.stringify(data.userinfo));
          }
        }, "json");
      };
    };
  </script>
```



创建实体类，用于关联二维码参数和 openid，如代码清单 10-2 所示。

代码清单 10-2

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace WxTest.CaseTwo
{
    public class WxLogin
    {
        /// <summary>
        /// 二维码参数值
        /// </summary>
        public int key { get; set; }
        /// <summary>
        /// 过期时间
        /// </summary>
        public DateTime exptime { get; set; }
        /// <summary>
        /// 扫描者的 openid
        /// </summary>
        public string openid { get; set; }
        /// <summary>
        /// 当前登录者列表
        /// </summary>
        private static List<WxLogin> _Logins;
        public static List<WxLogin> Logins
        {
            get
            {
                if (_Logins == null) _Logins = new List<WxLogin>();
                else
                {
                    _Logins.Where(l => l.exptime > DateTime.Now).ToList();
                }
                return _Logins;
            }
        }
    }
}
```



```
        set { _Logins = value; }  
    }  
}  
}
```

处理前台请求的后台代码如代码清单 10-3 所示。

代码清单 10-3

```
string accessToken = AccessTokenBox.GetTokenValue("appid", "appsecret");  
protected void Page_Load(object sender, EventArgs e)  
{  
    var action = Request.QueryString["action"];  
    //路由到对应的方法  
    switch (action)  
    {  
        case "loginqr": GetLoginQr(); break;  
        case "checkscan": CheckLogin(); break;  
    }  
}  
/// <summary>  
/// 获取登录二维码  
/// </summary>  
void GetLoginQr()  
{  
    var key = Utils.ConvertDateTimeInt(DateTime.Now);  
    var ticket = QrCode.CreateTemp(60, key, accessToken);  
    WxLogin.Logins.Add(new WxLogin { key = key,  
        exptime = DateTime.Now.AddMinutes(5) });  
    Application.Lock();  
    Application["logins"] = WxLogin.Logins;  
    Application.Unlock();  
    Response.Write(JsonConvert.SerializeObject(new { qrurl  
        = QrCode.GetQrByTicket(ticket.ticket), key = key }));  
    Response.End();  
}  
/// <summary>  
/// 检查登录状态  
/// </summary>
```




```
void CheckLogin()
{
    var key = Request.QueryString["key"];
    var info = WxLogin.Logins.FirstOrDefault(l => l.key.ToString() == key
    && !string.IsNullOrEmpty(l.openid));
    if (info != null)
    {
        var userinfo = WxApi.UserManager.BaseUser.GetUserInfo(info.openid,
        accessToken);
        #region 用户扫描成功后, 发送模板消息
        var dic = new Dictionary<string, TemplateKey>();
        dic.Add("first", new TemplateKey { color = "#173177", value = "您进行了
        微信扫一扫登录操作" });
        dic.Add("keyword1", new TemplateKey { color = "#173177", value =
        userinfo.nickname });
        dic.Add("keyword2", new TemplateKey { color = "#173177", value = "微企
        卡微信测试系统" });
        dic.Add("keyword3", new TemplateKey { color = "#173177", value =
        DateTime.Now.ToString() });
        dic.Add("remark", new TemplateKey { color = "#173177", value = "如有疑
        问, 请致电 IT 服务台 400-888-8888 或直接发送问题到公众号。谢谢。" });
        var templateid="uht719E3riw32NmF418cpRKh8mLWqjNqj-bGFsmGSls";
        TemplateNotice.Send(userinfo.openid, templateid, "#FF0000", dic,
        accessToken);
        #endregion
        Response.Write(JsonConvert.SerializeObject(new { status = 1, userinfo
        = userinfo }));
        Response.End();
    }
    else
    {
        Response.Write("{\"status\":0}");
        Response.End();
    }
}
```

用户扫描二维码后, 如果用户已经关注, 则推送的是扫描二维码事件的 XML 数据包; 否则推送的是关注事件的数据包。所以, 需要分别在关注事件和扫描带参数二维码事件中



处理扫描的结果。根据扫描的二维码的 key 值从数据库中或共享域中查找此二维码是否有记录,如果有则将扫描者的 openid 赋值给对象的 openid,当 PC 端获取到对应的 key 对象的 openid 属性为非空时,则说明用户已经扫描了,进而可以根据 openid 获取到用户信息,至此登录完成。

处理用户扫描二维码的事件代码如代码清单 10-4 所示。

代码清单 10-4

```
void QrLogin(BaseMsg baseMsg, string key)
{
    var app = HttpContext.Current.Application;
    app.Lock();
    var logins = app["logins"] as List<WxLogin>;
    var logininfo = logins.FirstOrDefault(l => l.key.ToString() == key);
    if (logininfo == null)
    {
        baseMsg.ResText(param, "不存在登录信息");
    }
    else
    {
        logininfo.openid = baseMsg.FromUserName;
        HttpContext.Current.Response.Write("");
    }
    app.UnLock();
}
```

只需要在处理关注事件和扫描带参数二维码的事件中调用上述方法即可。

10.2 网页分享——我是人气王

作为一个微信开发者,很多人每天打开最多的手机 App 应该就是微信了吧。相信大家或多或少收到过妹子们(可能部分男士也玩这个)分享的活动:就是心急火燎地让咱们帮忙助力,活动内容不是攒够多少助力送话费,就是送化妆品之类的。这对于咱们“收入分分钟上千万”的高端程序来说简直不屑一顾;不过为了讨妹子欢心,还是要“屁颠屁颠”地助力。闲话少说,本节就让我们自己动手做一个也让妹子激动的活动吧。



我是人气王，顾名思义，就是看谁的人气多。实现原理是：利用 JS-SDK 分享接口将用户分享的链接加一个和分享人关联的标示（openid 或数据库中的 userid），当分享的链接被别人打开后，使用 OAuth 授权方式获取访问者的用户信息，由于访问者访问的链接包括分析者的标示，所以当访问者单击“助力”按钮后，即可给对应的分享者的人气值加一。

详细的实现过程如下：

(1) 新建基类 WxBasePage，并继承 System.Web.UI.Page，此类作为所有需要使用微信网页授权的页面的基类。在此类的构造函数中绑定页面初始化事件，当页面初始化时，判断 Session 中是否保存了用户的信息。如果未保存则判断 code 是否为空，如果不为空则根据 code 获取用户信息；否则跳转到授权页面重新授权，如代码清单 10-5 所示。

代码清单 10-5

```
using WxApi;
using WxApi.ReceiveEntity;
using WxApi.UserManager;
namespace WxTest
{
    public class WxBasePage : System.Web.UI.Page
    {
        public const string appid = "appid";
        public const string appsecret = "appsecret";
        public string currenthost = Utils.GetCurrentFullHost();
        public string accessToken
        {
            get { return AccessTokenBox.GetTokenValue(appid, appsecret); }
        }
        /// <summary>
        /// 用户信息
        /// </summary>
        public UserInfo currentinfo
        {
            get { return (UserInfo)Session["userinfo"]; }
        }
        public OAuthToken AuthToken
        {
            get { return (OAuthToken)Session["AuthToken"]; }
        }
    }
}
```



```
}
public WxBasePage()
{
    //页面初始化事件
    this.Init += (s, e) =>
    {
        if (Session["userinfo"] == null)
        {
            var code = Request.QueryString["code"];
            if (string.IsNullOrEmpty(code))
            {
                Response.Redirect(OAuth.GetAuthUrl(appid,
GetRawUrl(), "1", AuthType.snsapi_userinfo));
            }
            else
            {
                Session["AuthToken"] = OAuth.GetAuthToken(appid,
appsecret, code);
                Session["userinfo"] =
OAuth.GetUserInfo(AuthToken.access_token, AuthToken.openid);
            }
        }
    };
}
/// <summary>
/// 获取当前请求的 URL
/// </summary>
/// <returns></returns>
public string GetRawUrl()
{
    return string.Format("http://{0}{1}", Utils.GetCurrentFullHost(),
HttpContext.Current.Request.RawUrl);
}
}
```

(2) 新建数据库，用户保存用户信息和助力记录。数据字典如表 10-1 所示。



表 10-1

Users (用户表)		
字段名	类 型	说 明
Id	int	主键, 自增
openid	varchar(50)	用户标示
NickName	nvarchar(50)	昵称
Headimgurl	varchar(200)	用户头像地址
Popular	int	人气值
Record (助力记录表)		
字段名	类 型	说 明
Id	int	主键, 自增
ShareOpenid	varchar(50)	分享者 openid
HelperOpenid	varchar(50)	助力者 openid
CreateTime	datetime	操作时间

(3) 新建页面 PopularKing。页面运行的效果如图 10-2 所示。



图 10-2

在页面中需要使用 JS-SDK 设置分享的内容, 包括图片、URL 和标题 (有关 JS-SDK 的使用请参考第 6 章)。分享的 URL 要带上当前用户的 openid 或者存储在数据库中的用户唯一标识, 形如 [http:// abc.com/PopularKing.aspx?shareid=openid](http://abc.com/PopularKing.aspx?shareid=openid), 如图 10-3 所示为配置分享内容



```

wx.config({
  debug: false, // 开启调试模式,调用的所有api的返回值会在客户端alert出来。若要查看传入的参数
  // 可以在pc端打开,参数信息会通过log打出,仅在pc端时才会打印
  appId: '<%=appid%>', // 必填,公众号的唯一标识
  timestamp: '<%=timestamp%>', // 必填,生成签名的时间戳
  nonceStr: '<%=noncestr%>', // 必填,生成签名的随机串
  signature: '<%=sign%>', // 必填,签名
  jsApiList: [
    'onMenuShareTimeline',
    'onMenuShareAppMessage', 'showOptionsMenu'
  ] // 必填,需要使用的JS接口列表
});
wx.ready(function() {
  wx.showOptionsMenu();
  var title = '我是人气王,我已经有<%=currentdr["Popular"]%>人气了,快来帮我攒人气吧.';
  var link = '<%=shareurl%>';
  var imgUrl = '<%=shareimg%>';
  wx.onMenuShareTimeline({
    title: title, // 分享标题
    link: link, // 分享链接
    imgUrl: imgUrl, // 分享图标
    success: function () {
      // 用户确认分享后执行的回调函数
      alert("分享成功");
    },
    cancel: function () {
      alert("取消分享");
      // 用户取消分享后执行的回调函数
    }
  });
});

```

图 10-3

当单击“助力”按钮时,首先判断分享者的 openid 和当前用户的 openid 是否相等,如果相等则说明当前的页面是分享者自己打开的,则不允许助力。如果不相等,则发送 POST 请求到当前页面,服务器后台判断请求是否是 POST,如果是 POST 则说明是助力请求。助力按钮的事件代码如代码清单 10-6 所示。

代码清单 10-6

```

<script>
$(function() {
  $("#helperntn").click(function() {
    var helper = $("#helper").val();
    var sharer = $("#sharer").val();
    if (sharer==helper) {
      alert("您不能为自己助力,请分享给您的朋友帮你助力!");
      _system._guide(true); // 分享引导
      return;
    }
    $.post(location.href, function(data) {

```



```
        if (data == 1) {  
            alert("助力成功");  
            location.href = location.href;  
        } else {  
            alert("请勿重复助力哦");  
        }  
    });  
});  
})  
</script>
```

在页面的后台代码中，PopularKing 需要继承 WxBasePage，详细代码如代码清单 10-7 所示。

代码清单 10-7

```
using System;  
using System.Data;  
using WxApi;  
using WxApi.ReceiveEntity;  
using WxApi.UserManager;  
  
namespace WxTest.CaseOne  
{  
    public partial class PopularKing : WxBasePage  
    {  
        public DataRow sharedr; // 分享信息  
        public DataRow currentdr; // 当前用户信息  
        public string openid;  
        protected string timestamp; // 时间戳  
        protected string noncestr; // 随机字符串  
        protected string url; // 当前 URL  
        protected string sign; // 签名  
        protected string shareurl; // 分享 URL  
        protected string shareimg; // 分享图片  
        static JsApiTicket ticket;  
        private string shareid;  
        protected void Page_Load(object sender, EventArgs e)  
        {
```



```
shareid = Request.QueryString["shareid"];
if (Request.HttpMethod.ToLower() == "get")
{
    openid = currentinfo.openid;
    //判断数据库是否有此微信用户的信息, 如果没有则写入数据库
    var isexist = Convert.ToInt32(SQLHelper.ExcuteScalarSQL
("select count(1) from [dbo].[Users] where openid='" + currentinfo.openid
+ "'")) != 0;
    if (!isexist)
    {
        //如果不存在, 则将信息添加到数据库中
        SQLHelper.ExcuteSQL(
string.Format("insert Users(openid,NickName,Headimgurl,
Popular)values('{0}','{1}','{2}',0)", currentinfo.openid,currentinfo.nickname,
currentinfo.headimgurl));
    }
    //判断此页面是否是从别人分享的页面进来的
    if (string.IsNullOrEmpty(shareid))
    {
        shareid = currentinfo.openid;
    }
    //分享者信息
    sharedr =
        SQLHelper.GetTable(string.Format("select * from Users
where openid='{0}'", shareid)).Rows[0];
    //当前登录的用户信息
    currentdr =
        SQLHelper.GetTable(string.Format("select * from Users
where openid='{0}'", currentinfo.openid)).Rows[0];
    //用户分享时, 分享的 URL
    shareurl = OAuth.GetAuthUrl(appid,
        string.Format("http://{0}/CaseOne/PopularKing.aspx?
shareid={1}", currenthost, currentinfo.openid), "", AuthType.snsapi_userinfo);
    // //用户分享时, 分享的图片
    shareimg = string.Format("http://{0}/logo.png", currenthost);
    timestamp = Utils.ConvertDateTimeInt(DateTime.Now).ToString();
    noncestr = timestamp; //随机字符串也使用时间戳
```




```
url = GetRawUrl();
if (ticket == null || ticket.expires_time < DateTime.Now)
{
    ticket = JsApi.GetHsJsApiTicket(accessToken);
}
sign = JsApi.GetJsApiSign(noncestr, ticket.ticket, timestamp,
url);

//获取活动排名信息, 并绑定
reList.DataSource = SQLHelper.GetTable("select * from Users
where Popular>0 order by Popular desc");
reList.DataBind();
}
else
{
    //POST 请求, 说明是助力操作。 首先查询是否已经助力了。不能为同一个人
    //重复助力
    if (Convert.ToInt32(SQLHelper.ExcuteScalarSQL(string.Format
("select COUNT(1) from Record where ShareOpenid='{0}' and
HelperOpenid='{1}'", shareid, currentinfo.openid))) == 0)
    {
        //插入助力记录, 并更新用户助力次数。 实际项目中需要使用事务, 限于
        //篇幅, 此处只实现了业务逻辑
        SQLHelper.ExcuteSQL(string.Format("insert
Record(ShareOpenid,HelperOpenid)values('{0}','{1}');update Users set
Popular=Popular+1 where openid='{0}'", shareid, currentinfo.openid));
        Response.Write("1");
    }
    else
    {
        Response.Write("0");
    }
    Response.End();
}
}
}
```



10.3 共享用户收货地址

微信收货地址共享,是指用户在微信浏览器内打开网页,填写过地址后,后续可以免填写、支持快速选择,也可增加和编辑。此地址为用户属性,可在各商户的网页中共享使用。支持原生控件填写地址,地址数据会传递到商户。

和微信支付一样,地址共享是基于微信 JavaScript API 实现的,只能在微信内置浏览器中使用,其他浏览器调用无效。同时,需要微信 5.0 版本才能支持,所以在使用时,需要判断用户的微信版本。编辑并获取用户收货地址 `editAddress` 接口,在网页前端调用。

参数列表如表 10-2 所示。

表 10-2

参 数	是 否 必 填	说 明
appId	是	公众号 AppID
scope	是	填写“jsapi_address”,获得编辑地址权限
signType	是	签名方式,目前仅支持 SHA1
addrSign	是	签名,由各参数一起参与签名生成
timeStamp	是	时间戳
nonceStr	是	随机字符串

参与 `addrSign` 签名的字段包括: `appId`、`url`(调用 JavaScript API 的网页 URL)、`timestamp`、`noncestr`、`accessToken`(并非是公众平台接口的全局 `accessToken`;而是网页授权时,根据 `code` 获取的,请参考 4.3 节的相关内容)。另外,用网页授权接口, `scope` 的值可以是 `snsapi_base` 或 `snsapi_userinfo`,而不是前端 JS 传入的 `jsapi_address`。由于使用了网页授权接口,URL 是经过微信回调后打开的,因此参与签名 URL 必须带上 `code` 和 `state` 参数。建议使用方法动态获取当前页面的 URL。

`addrSign` 的签名生成方式和微信支付的签名类似,只不过 `addrSign` 使用的加密方式是 SHA1,而微信支付使用的是 MD5。为了方便调用,将参与签名的参数封装为实体类,如代码清单 10-8 所示。

代码清单 10-8

```
namespace WxApi.PayEntity
{
    public class EditAddress
```



```
{  
    public string appid { get; set; }  
    public string url { get; set; }  
    public int timestamp { get; set; }  
    public string noncestr { get; set; }  
    public string accesstoken { get; set; }  
}  
}
```

代码清单 10-9 就是生成 addrSign 的方法。

代码清单 10-9

```
public static string GetEditAddressSign(EditAddress edit)  
{  
    var dic = Utils.EntityToDictionary(edit);  
    var arr = dic.OrderBy(d => d.Key).Select(d => string.Format("{0}={1}",  
        d.Key, d.Value)).ToArray();  
    string stringA = string.Join("&", arr);  
    return FormsAuthentication.HashPasswordForStoringInConfigFile(stringA,  
        "SHA1");  
}
```

新建页面 EditAddress.aspx, 在页面中添加一个按钮和一个 label 标签。单击按钮拉取用户的地址列表, 用户选择后, 获取回调的地址信息。按钮的事件代码如代码清单 10-10 所示。

代码清单 10-10

```
<script>  
    $(function () {  
        $("button").click(function () {  
            WeixinJSBridge.invoke('editAddress', {  
                "appId": "<%=Edit.appid%>",  
                "scope": "jsapi_address",  
                "signType": "sha1",  
                "addrSign": "<%=sign%>",  
                "timeStamp": "<%=Edit.timestamp%>",  
                "nonceStr": "<%=Edit.noncestr%>",  
            }, function (res) {
```



```

        if (res.err_msg=="edit_address:ok") {
            $("label").text("地址: " + res.proviceFirstStageName +
                res.addressCitySecondStageName +
                res.addressCountiesThirdStageName +
                res.addressDetailInfo+" 姓名: "+res.userName+" 手
机: "+res.telNumber+" 邮编: "+res.addressPostalCode);
        }
    });
});
}
</script>

```

后台代码如代码清单 10-11 所示。

代码清单 10-11

```

using System;
using WxApi;

namespace WxTest.CaseThree
{
    public partial class EditAddress :WxBasePage
    {
        public WxApi.PayEntity.EditAddress Edit;
        public string sign;
        protected void Page_Load(object sender, EventArgs e)
        {
            Edit = new WxApi.PayEntity.EditAddress
            {
                appid =appid,
                accesstoken =AuthToken.access_token,
                noncestr =Utils.GetGuid(),
                timestamp = Utils.ConvertDateTimeInt(DateTime.Now),
                url = GetRawUrl()
            };
            sign = WxApi.Pay.GetEditAddressSign(Edit);
        }
    }
}

```

执行流程如图 10-4 所示。

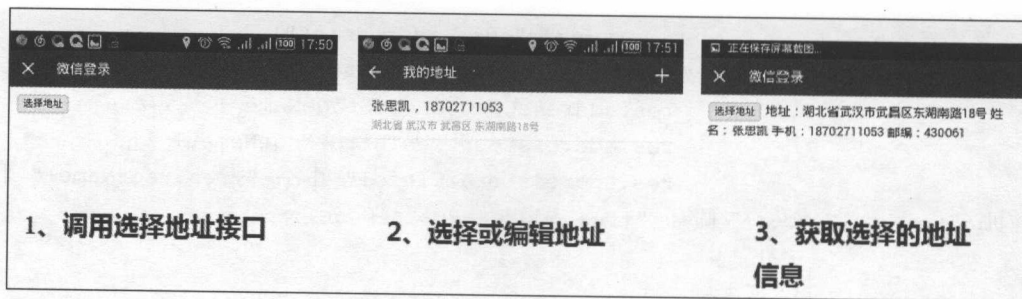


图 10-4

10.4 微信卡券应用——电影票

微信卡券的出现使公众号营销更上一层楼，简化了开发者的开发流程以及用户的体验。下面通过实际的案例来讲解卡券接口的调用。

看电影是生活中很常见的场景，中国电影市场动辄几亿、十几亿的票房也说明看电影的人越来越多了，而随之而来的就是售票窗口出现了长长的排队购票者。自主购票选座的必要性也越来越突出。本节将讲述微信卡券——电影票的用法。

首先，新建一个页面 `Movie.aspx`，用于创建电影票卡券。此页面的功能很简单，即生成一个电影票二维码，如代码清单 10-12 所示。

代码清单 10-12

```
var accessToken = AccessTokenBox.GetTokenValue("appid", "appsecret");
var info = new Movie_Ticket
{
    base_info = new BaseInfo
    {
        bind_openid = false,
        brand_name = "微企卡科技",
        can_give_friend = true,
        can_share = true,
        color = "Color010",
        code_type = CodeType.CODE_TYPE_QRCODE,
```



```
sku = new Sku { quantity = 50000000 },
date_info = new DateInfo
{
    type = 2,
    fixed_term = 30
},
description = "1.3 米（含）以下儿童在成人陪同下可免费观看 2D 电影（3D 影片无优惠，儿童单独观看 2D 影片无优惠），无座位，且每位成人限带 1 名儿童（以上仅供参考，具体以各门店为准），如遇动画片/儿童影片，优惠详情请咨询商家店面",
get_limit = 30,
logo_url = "http://mmbiz.qpic.cn/mmbiz/icIxxjVX9TaZe6jxWAQQ5ydyO79LGEfQYAvAzk4lN4VKKFmkD6P86ZvGFM3SmYlK0SNUAx9FS5GTicLf3vfp7A/0",
notice = "请出示二维码核销卡券",
promotion_url = "http://ypyle.xicp.net/CaseFour/Select.aspx",
promotion_url_sub_title = "点击进入",
promotion_url_name = "在线选座",
service_phone = "020-88888888",
title = "3D 电影家庭套票",
use_custom_code = false,
},
detail = "免费提供 3D 眼镜，无需押金，有部分影院是需要顾客自行购买 3D 眼镜的（每幅加 10 元） 具体见店内公告"
};

var ss = CardVoucher.Create(info, accessToken);
img.ImageUrl = QrCode.GetQrByTicket(CardVoucher.CreateCardQrTicket
(ss.card_id, accessToken).ticket);
```

运行此页面，页面上会显示一个二维码，扫描二维码后领取卡券，如图 10-5 所示。

在图 10-5 中，左图是领取后的卡券，单击“在线选座”按钮则进入中间的图所示的页面，选择座位号调用电影票更新接口，调用成功后，此张卡券则会从左图变成了右图的样子。在此单击“在线选座”按钮，可进入选座页面重新选座。

在选座页面中，单击每个座位时可更改座位的选中状态，单击座位的处理事件如代码清单 10-13 所示。



图 10-5

代码清单 10-13

```
$("#li").click(function () {  
    if (!$("this").hasClass("select") && $(".select").size() >= 4) {  
        alert("对不起, 您最多选择 4 个座位!"); //根据具体逻辑限制座位的数量  
        return;  
    }  
    $(this).toggleClass("select");  
});
```

单击“确定”按钮时, POST 请求当前页面, 并将选择的座位拼接为字符串提交到页面, 如代码清单 10-14 所示。

代码清单 10-14

```
$("#input[type=button]").click(function () {  
    var size = $(".select").size();  
    if (size != 4) {  
        alert("请选择 4 个座位");  
        return;  
    }  
});
```



```
var data = "";
$(".select").each(function () {
    data += $(this).text() + ",";
});
$.post(location.href, { data: data }, function (res) {
    if (res.ErrCode == 0)
    { alert("选择成功"); wx.closeWindow(); }
    else
    {
        alert(res.ErrDescription);
    }
}, "json");
})
```

在后台处理 POST 请求的代码中, 首先根据 encrypt_code 获取真实的卡券 code, 然后调用更新电影票接口, 如代码清单 10-15 所示。

代码清单 10-15

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Request.HttpMethod.ToLower() == "post")
    {
        var accessToken = AccessTokenBox.GetTokenValue("appid", "appsecret");
        //解码获取 code
        var code = CardVoucher.Decrypt(accessToken,
            Request.QueryString["encrypt_code"]);
        var v = CardVoucher.UpdateMovieTicket(accessToken, code.code, "3D",
            Utils.ConvertDateTimeInt(DateTime.Now.AddHours(2)),
            120, "5 号影厅",
            Request["data"].Split(new char[] { ',' },
            StringSplitOptions.RemoveEmptyEntries));
        Response.Write(JsonConvert.SerializeObject(v));
        Response.End();
    }
}
```


博文视点诚邀精锐作者加盟

《C++ Primer (中文版) (第5版)》、《淘宝技术这十年》、《代码大全》、《Windows 内核情景分析》、《加密与解密》、《编程之美》、《VC++ 深入详解》、《SEO 实战密码》、《PPT 演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉 Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与 IT 业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于 IT 专业出版之巅峰。

英雄帖

江湖风云起,代有才人出。

IT 界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享 IT 心得

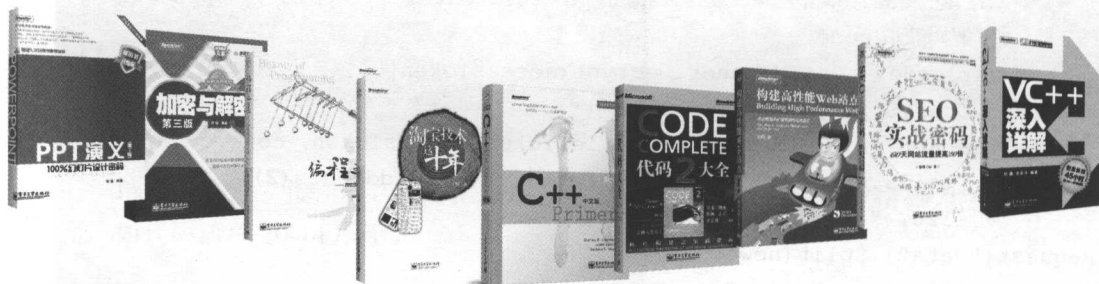
· 专业的作者服务 ·

博文视点自成立以来一直专注于 IT 专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照 IT 技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们

博文视点官网: <http://www.broadview.com.cn>

CSDN 官方博客: <http://blog.csdn.net/broadview2006/>

投稿电话: 010-51260888 88254368

投稿邮箱: jsj@phei.com.cn



@博文视点Broadview



微信公众账号

博文视点Broadview



博文视点精品图书展台

专业典藏



移动开发



大数据 · 云计算 · 物联网



数据库



Web 开发



程序设计



软件工程



办公精品



网络营销



博文视点本季最新最热图书



《淘宝技术这十年》

子柳 著

定价：45.00元

- ◎ 淘宝技术大学校长辛辣揭秘
- ◎ 世界最大电商平台、超大型网站，首次全面曝光技术内幕
- ◎ 技术变迁熠熠生辉、产品演进饱含智慧、牛人生涯叱咤风云、圈内趣事令人捧腹



《Windows内核原理与实现》

潘爱民 著

定价：99.00元

第一本用真实的源代码剖析 Windows 操作系统核心原理的原创著作！

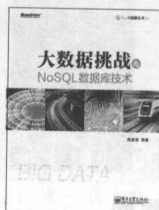


《软件需求最佳实践——SERU过程框架原理与应用（典藏版）》

徐锋 著

定价：69.00元

“用户说不清需求”、“需求变更频繁”……，都是在软件需求实践中频繁遇到的问题；本书首先直面这些问题，从心理学、社会学角度剖析其背后的深层原因，使大家从中获得突破的方法。



大数据丛书

《大数据挑战与NoSQL数据库技术》

陆嘉恒 编著

定价：79.00元

大数据技术的学习指南。突破迷局，厘清思路，拥抱变化。



《高性能MySQL（第3版）》

【美】施瓦茨（Schwartz,B.）【美】扎伊采夫（Zaitsev,P.）【美】特卡琴科（Tkachenko,V.）著
宁海元 周振兴 彭立勋等 译
定价：128.00元

本书是MySQL领域的经典之作，拥有广泛的影响力。第3版更新了大量的内容，不但涵盖了最新MySQL 5.5版本的新特性，也讲述了关于固态硬盘、高可扩展性设计和云计算环境下的数据库相关的新内容，原有的基准测试和性能优化部分也做了大量的扩展和补充。



《收获，不止Oracle》

梁敬彬 梁敬弘 著

定价：59.00元

颠覆IT技术图书的传统写作方式，在妙趣横生的故事中学到Oracle核心知识与优化方法论，让你摆脱技术束缚，超越技术。



《Clojure编程》

【美】Chas Emerick（蔡司·埃默里克）【美】Brian Carper（布赖恩·卡珀）【法】Christophe Grand（克里斯托弗·格兰德）著

徐明明 杨寿勋 译

定价：99.00元

第一本完整讲述Clojure的权威著作。



百度认证系列丛书

《百度推广——搜索营销新视角》

百度营销研究院 著

定价：59.00元

百度营销研究院资深专家团队撰写，百度认证初级教程！

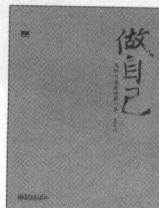


《我看电商》

黄若 著

定价：39.00元

作者近三十年从事零售及电子商务管理的总结和分享。



《做自己——鬼脚七自媒体第一季》

鬼脚七 著

定价：77.00元

本书是鬼脚七自媒体的原创文集，是电商圈第1本自媒体著作！书中有关于生活、互联网、自媒体的睿智分享，也有关于淘宝、搜索的独到见解。文章非常耐读，也容易引发读者的思考，是一本接地气、文艺范、充满正能量的电商生活书！

欢迎投稿：

投稿邮箱：jsj@phei.com.cn

editor@broadview.com.cn

读者信箱：market@broadview.com.cn

电话：010-51260888

更多信息请关注：

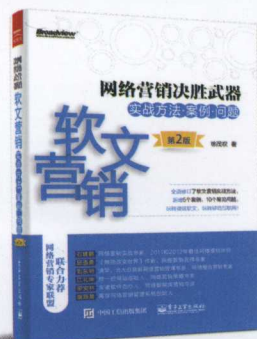
博文视点官方网站：

http://www.broadview.com.cn

博文视点官方微博：

http://t.sina.com.cn/broadviewbj

好书力荐



微信公众平台开发与案例分析

移动互联网时代，信息瞬息万变，微信公众号异军突起，众开发者也纷纷加入。本书来源于作者近三年微信公众平台开发经验的总结，以C#为技术基础，详细讲解微信公众平台的所有基础接口和绝大部分高级接口的调用、代码编写以及使用场景。从公众平台的工作原理到基础的开发与调试环境的搭建，再到基础服务接口的使用，最后在讲解各个高级接口调用的同时，结合实战案例，使读者对各个接口的调用以及使用场景有个充分的认识。

本书适合.NET平台下，有一定C#开发经验的开发者阅读，也可作为技术培训机构的参考教材。



博文视点Broadview



@博文视点Broadview



策划编辑：陈晓猛
责任编辑：李云静
封面设计：李玲

上架建议：微信开发

ISBN 978-7-121-27116-8



定价：79.00元(含DVD光盘1张)